Arduino bioreaktor Herbion

v0.9.0

Generated by Doxygen 1.12.0

Chapter 1

main issue

1.0.1 this project working by using platformio extension for visual studio code

1.0.2 Code is not fully tested

1.0.3 all code on develop branch is still in progress it is not recommend to copy/clone any part of it

1.1 Hardware Documentation

1.1.1 Check Hardware_configuration for more info

note that this documentation do not contain information about components connection

1.2 Hardware and software Configuration

1.2.1 Check Configuration for more info

how to setup environment and connect pin to make project work with base setup

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 arduino_task< T > Class Template Reference

Public Member Functions

- **arduino_task** (uint8_t priority, unsigned long repeat_time, T ∗object, void(T::∗method)())
- void **examine_task_state** (unsigned long time)
- void **run_task** ()

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

- arduino_os.hpp

5.2 Config_var Class Reference

Public Member Functions

- **Config_var** (float desire_ph, float max_ph_acceptable_deviation, float desire_temp, float max_temp_↩ acceptable_deviation)

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

- utility.hpp
- utility.cpp

5.3 ConfigurationVariable Class Reference

Public Member Functions

- **ConfigurationVariable** (MemoryManager &mem_manager)
- int **get_addr** () const
- float **return_config_value** () const
- void **retrieve_config_values_from_eeprom** ()
- void **change_config_value** (float value)

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

- eeprom_menager.hpp
- eeprom_menager.cpp

5.4 DataHMS Class Reference

Public Member Functions

- **DataHMS** (long hour, long minute, long second)
- String **return_data** ()

Public Attributes

- long **m_offset**

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

- utility.hpp
- utility.cpp

5.5 MeasureArray Class Reference

Public Member Functions

- **MeasureArray** (int size)
- void **init** (float initial_value)
- void **add_measure** (float value)
- float **read_measure** (char index)
- float **get_average** ()

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

- utility.hpp
- utility.cpp

5.6 MemoryManager Class Reference

Public Member Functions

- **MemoryManager** (int memory_start, int memory_length)
- int **give_memory** (int require_memory)

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

- eeprom_menager.hpp
- eeprom_menager.cpp

5.7 my_rotary_encoder Class Reference

Public Member Functions

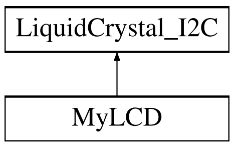
- **my_rotary_encoder** (uint8_t pin_a, uint8_t pin_b, uint8_t button, long sensitivity)
- void **init** ()
- int **get_encoder_pos** ()
- int **get_button_state** ()
- long **return_button_inactivate_state_time** ()
- void **check_encoder_pos** ()
- void **reset_encoder_pos** ()
- int **get_encoder_move** ()
- float **set_value** (float initial_value, float step, MyLCD lcd)
- int **get_button_depth** ()
- void **reset__button_depth** ()

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

- my_encoder.hpp
- my_encoder.cpp

5.8 MyLCD Class Reference

Inheritance diagram for MyLCD:



Public Member Functions

- **MyLCD** (uint8_t addr, uint8_t column_num, uint8_t row_num)
- void **initialize** ()
- void **send_float_value** (String text, float value, int lcd_row)
- void **send_string** (String text, String value, int lcd_row)

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

- lcd_display.hpp
- lcd_display.cpp

5.9 SdMemoryManager Class Reference

Main purpose is write data to SD card.

```
#include <sd_memory.hpp>
```

Public Member Functions

- **SdMemoryManager** (uint8_t mosi_pin, uint8_t miso_pin, uint8_t sck_pin, uint8_t cs_pin)
- void **init** ()
- void **write_to_st** ()
- void write_data_frame_to_st (Sensor &thermometer, Sensor &ph_meter, Sensor &oxygen_meter, DataHMS &data)
- String **DEBUG_write_data_frame** (Sensor &thermometer, Sensor &ph_meter, Sensor &oxygen_meter, DataHMS &data)
- void **save** ()
- void **close_file** ()

5.9.1 Detailed Description

Main purpose is write data to SD card.

this class mostly use function in SD.h library main idea is to create file and write data to it

5.9.2 Member Function Documentation

5.9.2.1 write_data_frame_to_st()

```
void SdMemoryManager::write_data_frame_to_st (
            Sensor & thermometer,
            Sensor & ph_meter,
            Sensor & oxygen_meter,
            DataHMS & data)
```

Parameters

thermometer	
ph_meter	
oxygen_meter	
data	

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

- sd_memory.hpp
- sd_memory.cpp

5.10 Sensor Class Reference

Public Member Functions

- **Sensor** (MeasuringDevice *measuring_dev, ConfigurationVariable &zero_shift, ConfigurationVariable &linear_factor)
- float **get_value_from_measurement** ()
- float **get_value** ()
- void **init** ()

Public Attributes

- ConfigurationVariable **m_zero_shift**
- ConfigurationVariable **m_linear_factor**
- MeasuringDevice ∗ **m_measuring_device**

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

- eeprom_menager.hpp
- eeprom_menager.cpp

5.11 TimerLowPriority Class Reference

Public Member Functions

- bool **activate** (int time_to_activate)
- void **reset** ()

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

- utility.hpp
- utility.cpp

Chapter 6

File Documentation

6.1 arduino_os.hpp

```
//some kind basic job schedule optimized to minimal size and ram usage


//general idea - adding task then planer will check

#include <Arduino.h>

#ifndef ARDUINO_OS
#define ARDUINO_OS

template <typename T>
class arduino_task {
private:

    const uint8_t m_priority;
    const unsigned long m_repeat_time;


    T* m_object;
    void (T::*m_task_method)();


    uint8_t m_state;
    unsigned long m_last_call;

public:

    arduino_task(uint8_t priority, unsigned long repeat_time, T* object, void (T::*method)());
    void examine_task_state(unsigned long time);
    void run_task();
};

template <typename T>
arduino_task<T>::arduino_task(uint8_t priority, unsigned long repeat_time, T* object, void
(T::*method)())
    : m_priority(priority),
      m_repeat_time(repeat_time),
      m_object(object),
      m_task_method(method),
      m_state(0),
      m_last_call(0)
{}

template <typename T>
void arduino_task<T>::examine_task_state(unsigned long time) {
    if (time >= m_last_call + m_repeat_time) {
        m_state = 1;
        m_last_call = time;
    } else {
        m_state = 0;
    }
}

template <typename T>
void arduino_task<T>::run_task() {

    if(m_state==1){(m_object->*m_task_method)();}
    m_state = 0;

}




#endif //ARDUINO_OS
```

6.2 bioreactor_defined_const.hpp

```
#ifndef BIOREACTOR_CONST
#define BIOREACTOR_CONST

#include<Arduino.h>

//BUTTON SETTING - CAN BE CONFIGURABLE!
const char BUTTON_DEFAULT_STATE = 1; //mean pin state is HIGH
const long int BUTTON_STAY_ON_STATE = 2000; //[ms] time button dont change its state
const long int STAY_IN_MENU_TIME = 3000; //[ms] time after you can exit menu


//ROTARY ENCODER SETTINGS - CAN BE CONFIGURABLE!

const unsigned long SENSITIVITY = 150; // [ms] time after re will not change it state after did it
before //TODO use better world :)


//LCD SETTINGS
// for now not available

//SENSOR DESIRE PARAMETER - CAN BE CONFIGURABLE!
const float DESIRE_PH = 7;
const float MAX_PH_ACCEPTABLE_DEVIATION = 0.5;
const float DESIRE_TEMP = 21.37;
const float MAX_TEMP_ACCEPTABLE_DEVIATION = 4.2069;


//LCD PHYSICAL CONNECTION
// TODO

//SD CARD READER PHYSICAL CONNECTION
constexpr uint8_t CS_PIN = 10;
constexpr uint8_t DEF_MOSI_PIN = 11;
constexpr uint8_t DEF_MISO_PIN = 12;
constexpr uint8_t DEF_SCK_PIN = 13;
#define FILENAME "m.txt"

//ROTARY ENCODER PHYSICAL CONNECTION
constexpr uint8_t RE_PIN_A = 3;
constexpr uint8_t RE_PIN_B = 4;
constexpr uint8_t RE_BUTTON_PIN = 5;

//SENSORS PHYSICAL CONNECTION
constexpr uint8_t PH_METER_PIN = A1;
constexpr uint8_t THERMOMETER_PIN = A2;
constexpr uint8_t OXYGEN_METER_PIN = A3;

//SAMPLE PUMP PARAMETER (this pump is peristaltic and work only in on/off mode)
constexpr int PUMP_FLOW_SPEED = 60; //[ml/min]
constexpr int SAMPLE_SIZE = 2; //[ml]
constexpr int PUMP_SAMPLE_TAKING_TIME = int(float(SAMPLE_SIZE)/float(PUMP_FLOW_SPEED)*60*1000);
//[ms]; should be 2000ms

//PH KEEPERS PUMP _PH_CORRECTION - CAN BE CONFIGURABLE!
const long int TIME_BETWEEN_PH_CORRECTION = 10000; //[ms] time between ph correction [TODO change
name]
const long int CORRECTION_TIME = 100; // [ms] initial time of correction
const float MULT_PER_DEGREE = 3; // it mean if  desire ph is 6 and we have 8 time of pump activate
will be 100*3*3 = 900 ms
const long int MAX_REACTION_TIME = 3000;  //max time of open the pump



// thermometer parameter (it is hardware parameter)
const long int THERMISTOR_NOMINAL = 10000;        // [ohm] resistance at 25 degrees C
const uint8_t TEMPERATURE_NOMINAL = 25;          // temp. for nominal resistance (almost always 25 C)
const long int B_COEFFICIENT = 3950;              // [UNIT!] The beta coefficient of the thermistor
(usually 3000-4000)
const long int SERIES_RESISTOR = 10000;           // [ohm] the value of the 'other' resistor



#endif // BIOREACTOR_CONST
```

6.3 components.hpp

```
#include <Arduino.h>
#include <math.h>
#include "bioreactor_defined_const.hpp"
#include "utility.hpp"

#ifndef BIOREACTOR_COMPONENTS
#define BIOREACTOR_COMPONENTS


class SimplePeristalticPump
{
private:
uint8_t m_pin_forward;


public:
SimplePeristalticPump(uint8_t pin_forward);
void init();
void stabilize_ph(float current_ph,float desire_ph);
};

class PeristalticPump
{
  private:
    const uint8_t m_pwm_pin;  //TODO snake case
    const uint8_t m_dir1_pin;
    const uint8_t m_dir2_pin;
    int m_current_pwm_value;

  public:
    PeristalticPump(uint8_t pwm_pin, uint8_t dir1_pin, uint8_t dir2_pin);  //TODO rename variable
    void init();
    void set_pump_speed(int value); // from -100 (max reverse speed) to 100 (max forward speed)
    long get_current_speed() const;
    void take_sample();
};

class MeasuringDevice
{
  //TODO: add virtual destructor even empty
  protected:
    const int m_read_pin;
    float m_value; // check what sensors return  TODO for now int ; but probably will be change in
inheritance

  public:
    MeasuringDevice(int read_pin);
    void init();
    virtual float get_value();
};

class Thermometer : public MeasuringDevice
{
  public:
    Thermometer(int read_pin);
    float get_value() override;
};

class PhMeter : public MeasuringDevice
{
  public:
    PhMeter(int read_pin);
    float get_value() override;
};

class OxygenMeter : public MeasuringDevice
{
  public:
    OxygenMeter(int read_pin);
    float get_value() override;
};

#endif // BIOREACTOR_COMPONENTS
```

6.4 eeprom_menager.hpp

```
#include "components.hpp"
#include <EEPROM.h>

#ifndef TEST
#define TEST
```

```

class MemoryManager
{
  private:
    const int m_memory_start;
    const int m_memory_size;
    int m_memory_pointer;

  public:
    MemoryManager(int memory_start, int memory_length);
    int give_memory(int require_memory);
};

class ConfigurationVariable
{
  private:
    int m_memory_addr;
    float m_value = -21.37;

  public:
    ConfigurationVariable(MemoryManager &mem_manager);
    int get_addr() const;
    float return_config_value() const;
    void retrieve_config_values_from_eeprom();
    void change_config_value(float value);
};

class Sensor
{
  private:
    float m_value = 10;

  public: // bad practice TODO in free time try to change to private
    ConfigurationVariable m_zero_shift, m_linear_factor; //TODO - make reference to this object not copy
    MeasuringDevice *m_measuring_device;

  public:
    Sensor(MeasuringDevice *measuring_dev, ConfigurationVariable &zero_shift, ConfigurationVariable &linear_factor);
    float get_value_from_measurement();
    float get_value();
    void init();
};

#endif // TEST
```

6.5 lcd_display.hpp

```
#ifndef LCD_DISPLAY_BR
#define LCD_DISPLAY_BR

#include<LiquidCrystal_I2C.h>
// not refactored

class MyLCD : public LiquidCrystal_I2C
{

  public:
  MyLCD(uint8_t addr, uint8_t column_num, uint8_t row_num);
  void initialize(); //TODO: better name
  void send_float_value(String text, float value, int lcd_row); //TODO: change to template
  void send_string(String text, String value, int lcd_row);
};

#endif // LCD_DISPLAY_BR
```

6.6 my_encoder.hpp

```
#ifndef MY_ENCODER
#define MY_ENCODER

#include <Arduino.h>
#include "lcd_display.hpp"


class my_rotary_encoder
{
  private:
    const uint8_t m_pin_a;
    const uint8_t m_pin_b;
    const uint8_t m_pin_button;

    uint8_t m_button_depth;
    unsigned long m_sensitivity;
    unsigned long m_button_inactivate_state_time; //TODO rename
    unsigned long m_last_change;

    uint8_t m_a_val = 0;
    uint8_t m_pinALast = 0;
    uint8_t m_button_state = 1;
    int m_encoderPosCount;

  public:
    my_rotary_encoder(uint8_t pin_a, uint8_t pin_b, uint8_t button, long sensitivity);
    void init();
    int get_encoder_pos();
    int get_button_state();
    long return_button_inactivate_state_time();
    void check_encoder_pos();
    void reset_encoder_pos();
    int get_encoder_move();
    float set_value(float initial_value, float step, MyLCD lcd);
    int get_button_depth();
    void reset__button_depth();
};

#endif
```

6.7 sd_memory.hpp

```
//TODO REFACTOR WHOLE CLASS in way which Define is not used in CPP file

#ifndef SM_MEMORY
#define SM_MEMORY

#include <SD.h>
#include "eeprom_menager.hpp"
#include "utility.hpp"

class SdMemoryManager
{
  private:
    // pins
    const uint8_t m_mosi_pin;
    const uint8_t m_miso_pin;
    const uint8_t m_sck_pin;
    const uint8_t m_cs_pin;

    uint8_t m_write_number=0;

    File m_file;
    String m_filename;
    String m_dataBuffer;
    unsigned long m_last_write = 0;

  public:

    SdMemoryManager(uint8_t mosi_pin, uint8_t miso_pin, uint8_t sck_pin, uint8_t cs_pin);

    void init();
    void write_to_st();
    void write_data_frame_to_st(Sensor& thermometer, Sensor& ph_meter, Sensor& oxygen_meter, DataHMS&
data);
    String DEBUG_write_data_frame(Sensor& thermometer, Sensor& ph_meter, Sensor& oxygen_meter,
DataHMS& data);
    void save();
    void close_file();
};

#endif

/*

#include <SD.h>

const int chipSelect = 10;

// file name to use for writing
const char filename[] = "datalog.txt";

// File object to represent file
```

```
File myFile;
// string to buffer output
String dataBuffer;
// last time data was written to card:
unsigned long lastMillis = 0;

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  // reserve 1 kB for String used as a dataBuffer
  dataBuffer.reserve(1024);

  // set LED pin to output, used to blink when writing
  pinMode(LED_BUILTIN, OUTPUT);

  // wait for Serial Monitor to connect. Needed for native USB port boards only:
  while (!Serial);

  Serial.print("Initializing SD card...");

  if (!SD.begin(chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("1. is a card inserted?");
    Serial.println("2. is your wiring correct?");
    Serial.println("3. did you change the chipSelect pin to match your shield or module?");
    Serial.println("Note: press reset button on the board and reopen this Serial Monitor after fixing
your issue!");
    while (true);
  }

  Serial.println("initialization done.");

  // If you want to start from an empty file,
  // uncomment the next line:
  //  SD.remove(filename);
  // try to open the file for writing

  myFile = SD.open(filename, FILE_WRITE);
  if (!myFile) {
    Serial.print("error opening ");
    Serial.println(filename);
    while (true);
  }

  // add some new lines to start
  myFile.println();
  myFile.println("Hello World!");
  Serial.println("Starting to write to file...");
}

void loop() {
  // check if it's been over 10 ms since the last line added
  unsigned long now = millis();
  if ((now - lastMillis) >= 10) {
    // add a new line to the dataBuffer
    dataBuffer += "Hello ";
    dataBuffer += now;
    dataBuffer += "\r\n";
    // print the buffer length. This will change depending on when
    // data is actually written to the SD card file:
    Serial.print("Unsaved data buffer length (in bytes): ");
    Serial.println(dataBuffer.length());
    // note the time that the last line was added to the string
    lastMillis = now;
  }

  // check if the SD card is available to write data without blocking
  // and if the dataBuffered data is enough for the full chunk size
  unsigned int chunkSize = myFile.availableForWrite();
  if (chunkSize && dataBuffer.length() >= chunkSize) {
    // write to file and blink LED
    digitalWrite(LED_BUILTIN, HIGH);
    myFile.write(dataBuffer.c_str(), chunkSize);
    digitalWrite(LED_BUILTIN, LOW);
    // remove written data from dataBuffer
    dataBuffer.remove(0, chunkSize);
  }
}

*/
```

6.8 sensor_config.hpp

```
#include "bioreactor_defined_const.hpp"
```

```
#include "components.hpp"
#include "eeprom_menager.hpp"
#include <Arduino.h>

#ifndef MY_SC
#define MY_SC

Sensor setup_thermometer_sensors(MemoryManager &manager);
Sensor setup_ph_sensors(MemoryManager &manager);
Sensor setup_oxygen_sensors(MemoryManager &manager);
void test_sensor(Sensor &sensor, float value_new, String sensor_name = "Sensor");

#endif
```

6.9 Unit_tests.hpp

```
#include "utility.hpp"
#include <AUnit.h>

#ifndef MY_UNIT_TEST
#define MY_UNIT_TEST

// not refactored

int dumb_func()
{
    return 1;
}

MeasureArray m1(10), m2(3); // 10 ; 3

float test_mes_array(MeasureArray &ma)
{
    ma.add_measure(20);
    ma.add_measure(10);
    ma.add_measure(5);
    ma.add_measure(5);
    ma.add_measure(60);
    return (ma.get_average());
}

float test_mes_array2(MeasureArray &ma)
{
    ma.add_measure(20);
    ma.add_measure(10);
    ma.add_measure(2137);

    return ma.read_measure(2);
}

test(test_mes_arrayTest)
{
    assertEqual(test_mes_array(m1), 10.0);
}

test(test_mes_array2Test)
{
    assertEqual(test_mes_array2(m2), 2137.0);
}

test(dumb_funcTest)
{
    assertEqual(dumb_func(), 1);
}

#endif // MY_UNIT_TEST
```

6.10 utility.hpp

```
#ifndef MY_UTILITY
#define MY_UTILITY

#include<string.h>
#include<Arduino.h>

#include "my_encoder.hpp"
#include "eeprom_menager.hpp"
#include "lcd_display.hpp"

```

```
class Config_var   //more like struct but... TODO check if we can change it to real struct
{
  private:
  float m_desire_ph;
  float m_max_ph_acceptable_deviation;

  float m_desire_temp;
  float m_max_temp_acceptable_deviation;

  public:
  Config_var(float desire_ph, float max_ph_acceptable_deviation, float desire_temp, float
max_temp_acceptable_deviation);
};




class MeasureArray // TODO is not good enough make it better
{
  private: //TODO: inheritance after arduino vector
  //TODO: is not good enough make it better
    const char m_array_size; // max 256 (but not recommended to use more than 100)
    float *measurement;      // Pointer array, will be initialized in constructor
    int m_memory_cursor = 0;
    float m_oldest_measure;

  public:
    MeasureArray(int size);
    void init(float initial_value); //TODO make it vector
    void add_measure(float value);
    float read_measure(char index);
    float get_average(); //TODO: add const
};

class TimerLowPriority
{
  private:
    unsigned long m_start_time;
    unsigned long m_end_time;

  public:
    TimerLowPriority();
    bool activate(int time_to_activate);
    void reset();
};

// TODO high priority timer

class DataHMS
{
  public:
    long m_offset;

  public:
    DataHMS(long hour, long minute, long second);
    String return_data(); //TODO const
};

void print_config_menu(my_rotary_encoder &encoder, MyLCD &lcd, // main sensors
                       Sensor &term, Sensor &ph, Sensor &oxygen, PeristalticPump& pump);

#endif
```

Index