Communication

Not peer-reviewed version

Open-Source Code: Data Logger System for Monitoring and Fault Detection in Bench Testing of Combustion Engines

Marcio Luís Munhoz Amorim *, Jorge Gomes Lima, Norah Nadia Sánchez Torres, Fabiano Salvadori, Oswaldo Hideo Ando Junior *

Posted Date: 19 June 2024

doi: 10.20944/preprints202406.1329.v1

Keywords: Open-Source Code; Data Logger; Combustion Engines; Non-Invasive Sensors; Open-Source System; Monitoring and Fault Detection

Communication

Open-Source Code: Data Logger System for Monitoring and Fault Detection in Bench Testing of Combustion Engines

Marcio Luís Munhoz Amorim [1]*, Jorge Gomes Lima [2], Norah Nadia Sánchez Torres [3], Fabiano Salvadori [2] and Oswaldo Hideo Ando Junior [2-5]*

[1] Group of Metamaterials Microwaves and Optics (GMeta), Department of Electrical Engineering (SEL), University of São Paulo (USP), Avenida Trabalhador São-Carlense, Nr. 400, Parque Industrial Arnold Schimidt, São Carlos 13566-590, Brazil; marciolma@usp.br (M.L.M.A.)

[2] Smart Grid Laboratory (LabREI), Center for Alternative and Renewable Research (CEAR), Federal University of Paraiba (UFPB), João Pessoa 58051-900, PB, Brazil; jorgeg.lima@cear.ufpb.br (J.G.L.); salvadori.fabiano@cear.ufpb.br (F.S).

[3] Research Group on Energy & Energy Sustainability (GPEnSE), Academic Unit of Cabo de Santo Agostinho (UACSA), Federal Rural University of Pernambuco (UFRPE), Cabo de Santo Agostinho 54518-430, PE, Brazil; oswaldo.ando@ufrpe.br (O.H.A.J.)

[4] Interdisciplinary Postgraduate Program in Energy & Sustainability (PPGIES), Federal University of Latin American Integration – UNILA, Paraná City, Brazil, 85.867-000. norah.torres@aluno.unila.edu.br (N.N.T.S.)

[5] Program in Energy Systems Engineering (PPGESE) Academic Unit of Cabo de Santo Agostinho (UACSA), Federal Rural University of Pernambuco (UFRPE), Cabo de Santo Agostinho 54518-430, PE, Brazil; oswaldo.ando@ufrpe.br (O.H.A.J.)

* Correspondence: marciolma@usp.br (M.L.M.A.); oswaldo.ando@ufrpe.br (O.H.A.J.)

Abstract: This paper describes the proposal and development of a proof of concept (PoC) for a data logger system designed for operation and fault detection in bench testing of combustion engines. The main objective was to develop a data logger system that collects data on vibration, sound, temperature, and CO2 levels to identify ignition and combustion faults, thereby contributing to reducing greenhouse gas emissions (GHG). The paper presents (i) the proposal and development and (ii) the open-source code: proof of concept for a data logger system focused on detecting faults and monitoring the operational condition of benchtop combustion engines. The results indicate the feasibility of the PoC for data acquisition and dataset creation for Research and Development purposes.

Keywords: open-source code; data logger; combustion engines; non-invasive sensors; open-source system; monitoring and fault detection

1. Introduction

Logistics plays a crucial role in the global economy, accounting for approximately 90% of international trade. According to the National Logistics and Transportation Plan of 2015, there is a projected 92% increase in cargo throughput at Brazilian ports by 2042, totaling 1.8 billion tons [1]. The Economic Commission for Latin America and the Caribbean and the World Population Review indicate that global annual greenhouse gas (GHG) emissions have increased by 41% since 1990, with the energy sector responsible for 73% of these emissions, 15% of which originate from the transportation sector [2,3]. This scenario underscores the urgent need for adopting sustainable technologies in maritime transport, aligned with the Sustainable Development Goals (SDGs) of the 2030 Agenda and the commitments of the Paris Agreement to achieve CO2 neutrality by 2050 [4]. Brazil has committed to reducing its CO2 emissions by 37% by 2025, with a target of 43% reduction by 2030 [5,6].

 doi:10.20944/preprints202406.1329.v1

Maritime transport is fundamental to global trade, accounting for over 90% of goods transportation. However, the sector faces significant challenges regarding air pollution, especially in coastal areas. CO2 emissions from maritime activities represent approximately 3% of the global total, with projections indicating a potential increase of up to 250% by 2050. In response to this situation, the International Maritime Organization (IMO) has set targets to reduce the carbon intensity of ships and promote the adoption of zero-emission technologies by 2030 [7,8].

Given the strategic importance of the maritime industry to global trade and the sustainability imperatives proposed by the IMO, this project investigates the development of a device for monitoring the mechanical transmission system of vessels. This monitoring aims to enhance operational efficiency and reduce GHG emissions by optimizing the early detection of faults in internal combustion engines (ICEs), which can result from design, installation, maintenance, or operational errors [9,10].

In this context, this project aims to develop a naval telemetry platform for coastal waterway transport. The platform will consist of an embedded data acquisition system, comprising hardware and software, facilitating telemetry of engines and other vessel devices. This solution will integrate with a logistics center via IoT, cloud computing, and a real-time control dashboard. As part of the project's development, it is essential to conceive and develop a proof of concept (PoC) to validate the efficiency and technical feasibility of the telemetry system.

This paper presents the conception and development of the PoC to validate its functionality and applicability in detecting faults on bench combustion engines. The structure of the article is organized into three main sections, addressing different aspects of naval telemetry platform development. Section 1 contextualizes the project's objectives, emphasizing the relevance of telemetry in naval contexts. Section 2 details the materials and methods used, including the specification and implementation of the PoC, along with preliminary results on the technical feasibility of applying the PoC for fault detection and monitoring of combustion engines in bench testing. Section 3 discusses key considerations for future research, exploring potential PoC improvements.

2. Materials And Methods

This section details the procedures used to develop and implement the open-source data logging system for monitoring and fault detection in bench testing of combustion engines. It describes the process of designing and assembling the data acquisition system for combustion engines.

2.1. Proof of Concept (PoC): Details and Specification

For the development of the project, considering the use of a low-cost and low computational effort methodology, and applying non-invasive techniques, there is the possibility of using Digital Signal Processors (DSP). Based on the state of the art, an ESP32 microcontroller board equipped with embedded Wi-Fi was selected. Initially, the types of data to be collected were defined for telemetry system development: vibration, sound, temperature, CO2 level, and organic compounds.

After analyzing the required data, compatible sensors and modules available in the market providing accurate information were chosen. Subsequently, the prototype creation process for testing began. The first step involved the development of a schematic diagram containing the ESP32, modules, and sensors. Components were soldered onto a busboard PCB, including a battery for operational support. PLA filament 3D printing was used to produce fittings for accommodating the PCB and battery. These fittings were housed within a rectangular aluminum profile to maintain system stability in the application environment.

For the initial test, code was written to control the modules and sensors using the Arduino IDE interface. Figure 1 depicts the 3D rendering of the project, illustrating (a) the device's operating screen, (b) control buttons, (c) CO2 and organic compound sensors, (d) USB input for ESP32 analysis and reprogramming, positioned above the temperature sensor, (e) microphone, and (f) MicroSD card slot.

 doi:10.20944/preprints202406.1329.v1

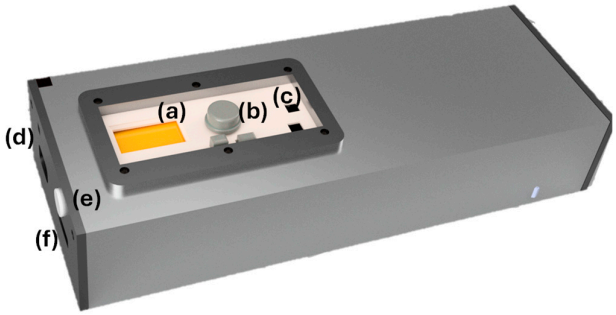

Figure 1. - Demonstration of the 3D PoC project.

2.2. Proof of Concept (PoC): Hardware

This section presents the prototype of the proof of concept (PoC) for the data acquisition system focused on monitoring and fault detection in bench testing of combustion engines. Structurally, the PoC includes a perforated PCB for component accommodation, AMG20 wires, a battery charging module, an ON/OFF button, PLA filament for 3D printing of the part that houses the PCB and modules, and an aluminum profile to assemble all components into a single device.

The Figure 2 showcases the developed prototype as a proof of concept (PoC) for a data acquisition system focused on detecting faults and monitoring the operational condition of combustion engines in bench testing. In the Figure 3 illustrates the structural components of the PoC. Figure 3.a depicts the aluminum profile with cutouts, finishings, and screws to accommodate the PCB assembly (Figure 3b,c), battery assembly, and display with buttons (Figure 3d). These assemblies are integrated into the aluminum profile (Figure 3e), shown in the top view (Figure 3f), left side view (Figure 3g), and right side view (Figure 3h), respectively.

Figure 2. – Demonstration of the Final Prototype (PoC).

 doi:10.20944/preprints202406.1329.v1

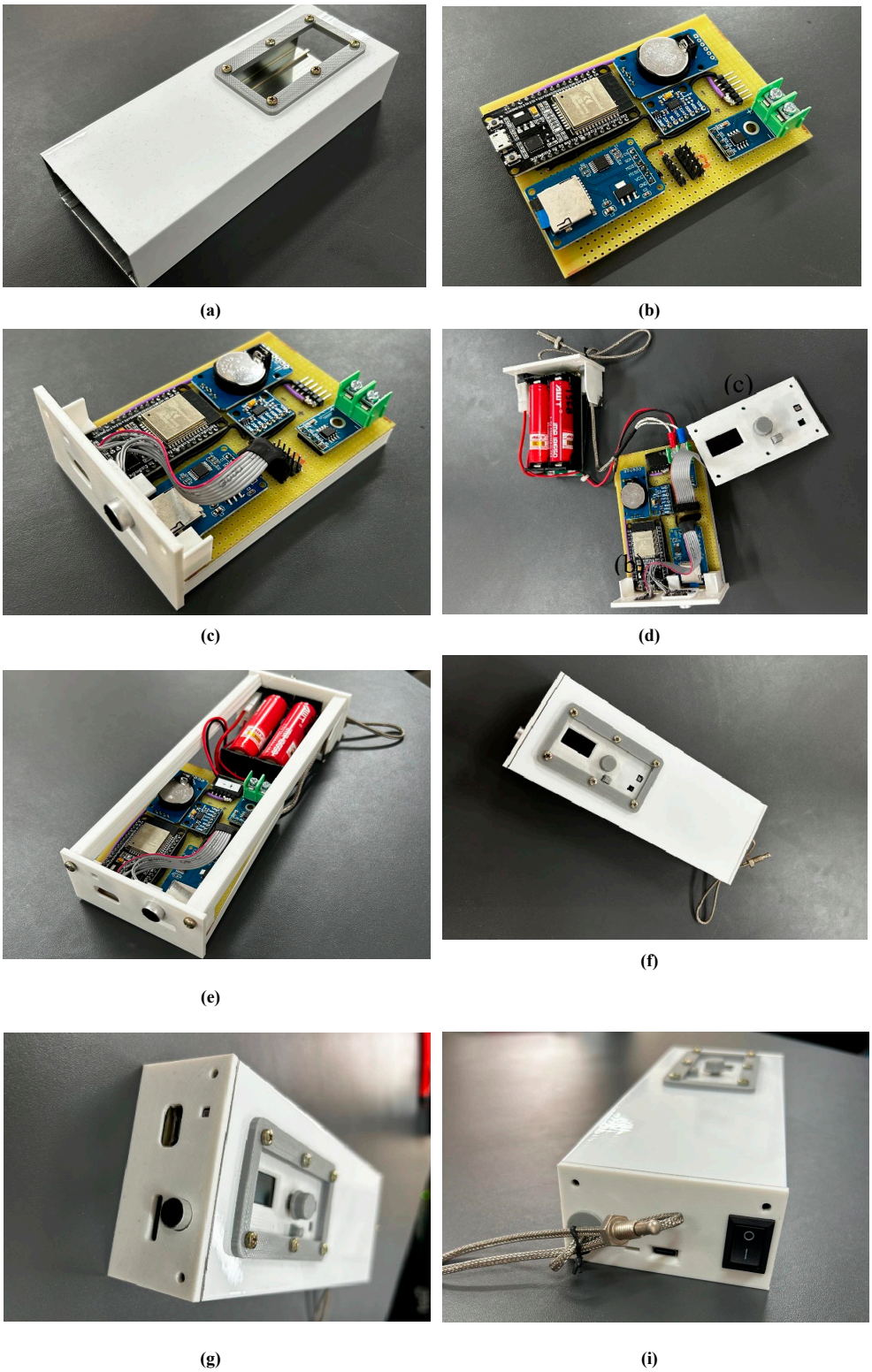

Figure 3. – Overview of the structural components and parts of the PoC.

2.3. Proof of Concept (PoC): Programming Code

The microcontroller programming was conducted using the Arduino IDE interface. The objective was to integrate the code for each sensor and module, enabling the prototype to capture information from all sensors simultaneously during testing. Before conducting comprehensive

 doi:10.20944/preprints202406.1329.v1

prototype tests with all sensors, it is essential to evaluate the functionality of each sensor individually. This ensures the optimization of code adjustments or component changes if any data deviates from expected parameters. The code developed and implemented in the data acquisition system is available in Appendix A.

3. Results and Conclusions

The proof of concept (PoC) implementation of a data acquisition system designed for detecting faults and monitoring the operational status of combustion engines during bench testing. The experiments utilized an ESP32 microcontroller along with various non-invasive sensors within a controlled laboratory setting. Results demonstrate successful capture of critical parameters—vibration, sound, temperature, and CO2 levels—essential for identifying operational anomalies. Integration of the ESP32 with Wi-Fi facilitated continuous real-time data acquisition, enhancing analytical capabilities.

Initial laboratory tests validated the prototype's functionality, affirming its efficacy. Utilization of open-source technology and non-invasive sensors not only proved efficient but also cost-effective, crucial for broad application across different contexts.

The development and application of this PoC for benchtop data acquisition present a promising strategy for fault detection and operational monitoring of engines. The integration of non-invasive sensors and affordable components such as the ESP32 effectively facilitated data collection for engine performance analysis. Based on these initial findings, ongoing adjustments and enhancements are planned to meet research requirements and improve system performance.

Future studies may expand upon this approach by conducting tests under operational conditions of combustion engines. Furthermore, establishing a comprehensive dataset from collected data could support the advancement of non-invasive diagnostic methodologies and fault detection algorithms, fostering technological innovation. This paper provides initial validation results of the PoC, currently in the developmental phase.

Author Contributions: Conceptualization: M.L.M.A., J.G.L., N.N.T.S., F.S and O.H.A.J; methodology: M.L.M.A., J.G.L., N.N.T.S., F.S and O.H.A.J; validation: M.L.M.A., F.S and O.H.A.J; investigation and simulation: M.L.M.A. and J.G.L., writing—original draft preparation: M.L.M.A., J.G.L., N.N.T.S., F.S and O.H.A.J; writing—review and editing: M.L.M.A., J.G.L., N.N.T.S., F.S and O.H.A.J; project administration: F.S. and O.H.A.J; funding acquisition: F.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the FACEPE agency (Fundação de Amparo a Pesquisa de Pernambuco) throughout the project with references APQ-0616-9.25/21 and APQ-0642-9.25/22. O.H.A.J. and F.S. was funded by the Brazilian National Council for Scientific and Technological Development (CNPq), grant numbers 407531/2018- 1, 303293/2020-9, 405385/2022-6, 405350/2022-8 and 40666/2022-3, as well as the Program in Energy Systems Engineering (PPGESE) Academic Unit of Cabo de Santo Agostinho (UACSA), Federal Rural University of Pernambuco (UFRPE). N.N.T.S were funded by the Federal University of Latin American Integration (UNILA).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A - Open-Source Code

```
//////SPI uSD STUFF ////////////////

//MOSI 23

//MISO 19

//SCLK 18

//SS      5 #include "FS.h"

#include "SD.h" #include "SPI.h" byte RECORD = 0; char

uSDstr[10];
```

 doi:10.20944/preprints202406.1329.v1


```
////////I2C STUFF///////////
//I2C
//SDA 21
//SCL 22
#include
<Wire.h>

/////////OLED STUFF/////////

#include <Adafruit_GFX.h> #define
SSD1306_128_32
#include <Adafruit_SSD1306.h> Adafruit_SSD1306 display = Adafruit_SSD1306();
unsigned long previousMillis = 0; const long interval =
2500;

////////ENS160 STUFF////////
//https://github.com/DFRobot/DFRobot_ENS160 #include
<DFRobot_ENS160.h>
DFRobot_ENS160_I2C ENS160(&Wire, /*I2CAddr*/ 0x52); // for 0x52 disconect SDO and GND
uint8_t Status; uint8_t AQI; uint16_t TVOC; uint16_t
ECO2;

///////BME STUFF//////////
//https://github.com/DFRobot/DFRobot_BME280 #include
"DFRobot_BME280.h"
typedef DFRobot_BME280_IIC          BME;   // ******** use abbreviations instead of full names ********
/**IIC address is 0x77 when pin SDO is high
*/
/**IIC address is 0x76 when pin SDO is low */ BME    bme(&Wire, 0x76);      // select
TwoWire peripheral and set sensor address #define SEA_LEVEL_PRESSURE 1015.0f
float     temp
= bme.getTemperature();
uint32_t pres = bme.getPressure(); floatalti =
bme.calAltitude(SEA_LEVEL_PRESSURE, pres); float          humi = bme.getHumidity();

/////////JOY STUFF////////// #define btnUP 26
#define btnDWN 27
#define btnLFT 14 #define btnRGT
12
#define btnCTR 15
#define btnAUX 13 byte UP = 1;
byte DWN = 1;

byte LFT = 1; byte RGT
```

 doi:10.20944/preprints202406.1329.v1


```
= 1;
byte CTR = 1; byte AUX = 1; byte LATCH = 0;

////////ACELL/////////////// #include
<Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
Adafruit_ADXL345_Unified
accel = Adafruit_ADXL345_Unified(12345); float acellX = 0;
float acellY = 0; float acellZ =
0;

////////RTC////////////////// #include <RTClib.h> RTC_DS3231 rtc;
char daysOfWeek[7][12] = {
“Sunday”,
“Monday”, “Tuesday”,

“Wednesday”, “Thursday”, “Friday”,

“Saturday”
};
uint16_t Year = 0; uint16_t Month = 0; uint16_t Day = 0; uint16_t
Hour = 0; uint16_t Min = 0; uint16_t Sec = 0;

/////MAX6675 STUFF////////// #include
“max6675.h”
#define thermoDO 35
#define thermoCS 32
#define thermoCLK 33 MAX6675
thermocouple(thermoCLK, thermoCS, thermoDO); float TEMP = 0;

//////////MIC STUFF////////////////////////
// ADC D34
uint16_t ADC = 0;

///////////INT TO STRING/////////////////////
void intToChar(int num, char* buffer, size_t bufferSize) {

itoa(num, buffer, 10); // 10 is the base (decimal)
}
///////////UPTIME STUFF/////////////////
// long Day=0; int uMillis = 0; int uHour = 0; int uMinute =
0;
//int Second=0;
int HighMillis = 0; int Rollover = 0; int timex = 0;
```

 doi:10.20944/preprints202406.1329.v1


```
long secsUp; byte

setUPTIME = 0;

////////////////////////////////////
////////////// SEUP
/////////////////
////////////////////////////////////// void setup() {
////I2C///

Wire.begin(21, 22); // Initialize I2C communication

////OLED///

display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //INICIALIZA O DISPLAY COM ENDEREÇO I2C 0x3C
display.setTextColor(WHITE); //DEFINE A COR DO TEXTO WHITE/BLACK/ INVERSE display.setTextSize(1); //DEFINE O TAMANHO DA FONTE DO TEXTO
display.clearDisplay(); //LIMPA AS INFORMAÇÕES DO DISPLAY

///////BME////////////// bme.begin();

///////ENS160/////////////// ENS160.begin(); ENS160.setPWRMode(ENS160_STANDARD_MODE);

ENS160.setTempAndHum(/*temperature=*/25.0, /*humidity=*/50.0);

/////////ACELL////////// accel.begin(0x53); accel.setRange(ADXL345_RANGE_16_G);

//////////SERIAL////////// Serial.begin(115200);

///RTC///
// SETUP RTC MODULE

if (! rtc.begin()) {
Serial.println("RTC module is NOT found");

Serial.flush(); while (1);
}
// automatically sets the RTC to the date & time on PC this sketch was compiled
rtc.adjust(DateTime(F( DATE ), F( TIME )));
//Manual

//rtc.adjust(DateTime(2024, 2, 12, 13, 20, 0)); delay(2000);
///////uSD////////// while (!SD.begin(5)) {
display.clearDisplay(); display.setCursor(30, 12);
```

 doi:10.20944/preprints202406.1329.v1


```
display.print("CARTAO FALHOU"); display.display();
//return;
}

uint8_t cardType = SD.cardType(); while (cardType == CARD_NONE) {

display.clearDisplay(); display.setCursor(20, 12); display.print("CARTAO NAO
ENCONTRADO");

display.display();
//return;
}
uint64_t cardSize = SD.cardSize() / (1024 * 1024);
Serial.printf("SD Card Size: %lluMB\n", cardSize);
//init File.SD
writeFile(SD, "/dl1.txt", "\n");

///JOY///
pinMode(btnUP, INPUT_PULLUP); pinMode(btnDWN, INPUT_PULLUP); pinMode(btnLFT,
INPUT_PULLUP);
pinMode(btnRGT, INPUT_PULLUP); pinMode(btnCTR, INPUT_PULLUP);

pinMode(btnAUX, INPUT_PULLUP);
}
////////////////////////////////////////
////////////// LOOP
/////////////////
//////////////////////////////////// void loop() {
AUX =
digitalRead(btnAUX);
if ((AUX != 1) && (LATCH == 0)) {
Record(); uMillis =
millis();
delay(2000);
LATCH = 1;
RECORD = 1;
}
AUX =
digitalRead(btnAUX);
if ((AUX != 1) && (LATCH == 1)) {
Closing();

delay(2000);
```

 doi:10.20944/preprints202406.1329.v1


```
LATCH = 0;
RECORD = 0;
}

///ACELL/// sensors_event_t event;

accel.getEvent(&event);
acellX = event.acceleration.x; acellY =
event.acceleration.y;
acellZ = event.acceleration.z;

///ADC/// ADC =
analogRead(34);
//intToChar(ADC, ADCstr, sizeof(ADCstr));

///MAX6675/// TEMP =
thermocouple.readCelsius();

///JOY/// JOY();

///uSD RUN/// uSD();

///UPTIME/// UPtime();

///ENS///

ens();

///BME/// bmee();

///OLED +
RTC///
unsigned long currentMillis = millis(); if (currentMillis - previousMillis >=
interval) {
RTC();
oled();
previousMillis = currentMillis;
}

///Serial TAB///

serial();
}
```

 doi:10.20944/preprints202406.1329.v1


```
////////////////////////////////////
////////////////////////////////////

//COMPONENTES
////////////////////////////////////
////////////////////////////////////

////////////////////////////////////
// BME
void bmee() { temp =
bme.getTemperature();
pres = bme.getPressure(); alti =
bme.calAltitude(SEA_LEVEL_PRESSURE, pres); humi = bme.getHumidity();
}
void
printLastOperateStatus(BME::eStatus_t eStatus)
{
switch (eStatus) { case
BME::eStatusOK:        Serial.println("everything ok"); break; case BME::eStatusErr:
Serial.println("unknow error"); break; case BME::eStatusErrDeviceNotDetected:
Serial.println("device not detected"); break; case BME::eStatusErrParameter:
Serial.println("parameter error"); break; default: Serial.println("unknow
status"); break;
}
}

////////////////////////////////////
//ENS
void ens()
{

Status = ENS160.getENS160Status(); Serial.print("Sensor operating status :
");
Serial.println(Status);

/**
Get the air quality index Return
value: 1-Excellent, 2-Good, 3-Moderate, 4-Poor, 5-Unhealthy
*/
AQI = ENS160.getAQI();

Serial.print("Air quality index : "); Serial.println(AQI);

/**
```

 doi:10.20944/preprints202406.1329.v1


```
Get
TVOC concentration
Return value range: 0–65000, unit: ppb
*/ TVOC =
ENS160.getTVOC();
Serial.print("Concentration of total volatile organic compounds :

");



Serial.print(TVOC); Serial.println(" ppb");



/**
Get CO2
equivalent concentration calculated according to the detected data of VOCs and hydrogen (eCO2
– Equivalent CO2)
Return value range: 400–65000, unit: ppm Five levels:
Excellent(400 - 600), Good(600 - 800), Moderate(800 - 1000),
Poor(1000 - 1500), Unhealthy(> 1500)
*/
ECO2 = ENS160.getECO2();
Serial.print("Carbon
dioxide equivalent concentration : "); Serial.print(ECO2); Serial.println("
ppm");
}

//////////////////////////////////////
//JOY
void JOY() {
//        UP = digitalRead(btnUP);
//        DWN = digitalRead(btnDWN);
//        LFT = digitalRead(btnLFT);
//
RGT = digitalRead(btnRGT);
//        CTR = digitalRead(btnCTR);
//AUX = digitalRead(btnAUX);

//        if (UP !=1){
//        Serial.println("UP");
//        }
//        if (DWN !=1){
// Serial.println("DOWN");
//        }
```

 doi:10.20944/preprints202406.1329.v1


```
//          if (LFT !=1){
// Serial.println("LEFT");
//          }
//          if (RGT !=1){
// Serial.println("RIGHT");
//          }
//          if (CTR !=1){
// Serial.println("CENTER");
//
}
}

////////////////////////////////
//OLED
//128x64 pixel
// Each pixel has 8x8 pixels

void oled() { display.clearDisplay();
//////////////LINHA 1//////////

display.setCursor(0, 0); display.print(Year, DEC); display.setCursor(26, 0);

display.print('/'); display.setCursor(36, 0); display.print(Month, DEC);

display.setCursor(44, 0); display.print('/'); display.setCursor(52, 0);

display.print(Day, DEC); if (RECORD == 0) {
display.setCursor(72, 0);

display.print("Press OK");
}
if (RECORD == 1) {
display.setCursor(72,

0);



0);


display.print("@"); if (uHour <= 9) {
display.setCursor(88,

display.print("0"); display.setCursor(96, 0);
```

 doi:10.20944/preprints202406.1329.v1


```
display.print(uHour);

if (uMinute <= 9) { display.setCursor(104, 0);

display.print(":0"); display.setCursor(120, 0);

display.print(uMinute);
}
else {
display.setCursor(104, 0);

display.print(":");
display.setCursor(112, 0);

display.print(uMinute);
}




0);

}
if (uHour >= 10) { display.setCursor(88,

display.print(uHour); if (uMinute <= 9) {

display.setCursor(104, 0);
display.print(":0");

display.setCursor(120, 0);
display.print(uMinute);
}
else {

display.setCursor(104, 0);
display.print(":");

display.setCursor(112, 0);
display.print(uMinute);
}
}
}
```

 doi:10.20944/preprints202406.1329.v1


```
//////////////LINHA 2////////// display.setCursor(0, 8);

display.print("MIC:");

display.setCursor(28, 8); display.print(ADC);

display.setCursor(66, 8); display.print("CO2:"); display.setCursor(92, 8);

display.print(ECO2);
//////////////LINHA 3////////// display.setCursor(0, 16);

display.print("TMP1"); display.setCursor(30, 16); display.print(temp);

display.setCursor(66, 16); display.print("TMP2"); display.setCursor(96, 16);

display.print(TEMP);
//////////////LINHA 4////////// display.setCursor(0, 24);

display.print("X:"); display.setCursor(15, 24); display.print(acellX);

display.setCursor(40, 24); display.print("Y:"); display.setCursor(52, 24);

display.print(acellY); display.setCursor(85, 24); display.print("Z:");

display.setCursor(97, 24); display.print(acellZ); display.display();
}

void Record()
{
display.clearDisplay(); display.setCursor(50, 12);

display.print("WAIT"); display.display();
}
void Closing() {

display.clearDisplay(); display.setCursor(30, 12); display.print("FINISHING
UP");
display.display();
}

////////////////////////////////
//RTC
```

 doi:10.20944/preprints202406.1329.v1


```
void RTC()
{
DateTime now = rtc.now(); Year = now.year();
// Week = daysOfWeek[now.dayOfTheWeek()],
Month = now.month();
Day = now.day();
Hour =
now.hour();
Min = now.minute();
Sec =
now.second();
}

////////////////////////////
//SERIAL
void serial() {

Serial.print("Mic Level: "); Serial.println(ADC); Serial.print("Probe Temp:
"); Serial.println(TEMP);

Serial.print("RTC:");
Serial.print(Year,
DEC);
Serial.print('/'); Serial.print(Month, DEC); Serial.print('/');

Serial.print(Day, DEC);
Serial.print(" (");

//Serial.print(daysOfWeek[now.dayOfTheWeek()]);
//Serial.print(") ");

Serial.print(Hour, DEC); Serial.print(':'); Serial.print(Min, DEC);

Serial.print(':'); Serial.print(Sec, DEC); Serial.println(") ");

Serial.print("AXCEL:");
Serial.print("X: "); Serial.print(acellX); Serial.print("    ");
Serial.print("Y: "); Serial.print(acellY); Serial.print("    ");
Serial.print("Z: "); Serial.print(acellZ); Serial.println(" m/s^2 ");

Serial.print("temperature (unit Celsius): "); Serial.println(temp);
Serial.print("pressure (unit pa): "); Serial.println(pres);
Serial.print("altitude (unit meter):        "); Serial.println(alti);
Serial.print("humidity (unit percent):    "); Serial.println(humi);
```

 doi:10.20944/preprints202406.1329.v1


```
Serial.print("Sensor operating status : ");

Serial.println(Status); Serial.print("Air quality index : ");

Serial.println(AQI);
Serial.print("Concentration of total volatile organic compounds : ");
Serial.print(TVOC); Serial.println(" ppb");

Serial.print("Carbon dioxide equivalent concentration : "); Serial.print(ECO2);

Serial.println(" ppm");
Serial.println();
Serial.println();

Serial.println();
Serial.flush();
}

////////////////////////////
//UPTIME
void UPtime() {
//** Making Note of an expected rollover *****// if (millis() >=
3000000000) {

HighMillis = 1;
}
//** Making note of actual rollover **// if
(millis() <= 100000 && HighMillis == 1) { Rollover++;
HighMillis = 0;

}
secsUp = (millis() - uMillis) / 1000; uMinute = (secsUp / 60) % 60;
uHour = (secsUp
/ (60 * 60)) % 24;
}

////////////////////////////////
//uSD
//Functions:
//          listDir(SD, "/", 0);
//          createDir(SD, "/mydir");
//          listDir(SD, "/", 0);
//          removeDir(SD, "/mydir");
//          listDir(SD, "/", 2);
```

 doi:10.20944/preprints202406.1329.v1


```
//
writeFile(SD, "/hello.txt", "Hello ");
//        appendFile(SD, "/hello.txt", "World!\n");
//        readFile(SD, "/hello.txt");
//
deleteFile(SD, "/foo.txt");
//        renameFile(SD, "/hello.txt", "/foo.txt");
//        readFile(SD, "/foo.txt");
//        testFileIO(SD, "/test.txt");
//        Serial.printf("Total space: %lluMB\n", SD.totalBytes() / (1024 * 1024));
//        Serial.printf("Used space: %lluMB\n", SD.usedBytes() / (1024 * 1024));

void uSD() {
//Sequencia:
// DATA;TEMP ONBOARD;TEMP PROBE;MIC;CO2;BAR;ORG;UMID;X;Y;Z
if (RECORD == 1) {
////Grava Data////

intToChar(Year, uSDstr, sizeof(uSDstr)); appendFile(SD, "/dl1.txt", uSDstr);

appendFile(SD, "/dl1.txt", "/"); intToChar(Month, uSDstr,
sizeof(uSDstr));
appendFile(SD, "/dl1.txt", uSDstr); appendFile(SD,
"/dl1.txt", "/");
intToChar(Day, uSDstr, sizeof(uSDstr));

appendFile(SD, "/dl1.txt", uSDstr); appendFile(SD, "/dl1.txt",
";");
///Grava temperatura 1 e 2 intToChar(temp, uSDstr, sizeof(uSDstr));

appendFile(SD, "/dl1.txt", uSDstr); intToChar(TEMP, uSDstr,
sizeof(uSDstr));
appendFile(SD, "/dl1.txt", ";");
///Grava Micrifone e Nivel CO2
intToChar(ADC, uSDstr, sizeof(uSDstr)); appendFile(SD,
"/dl1.txt", uSDstr);

appendFile(SD, "/dl1.txt", ";");

intToChar(ECO2, uSDstr, sizeof(uSDstr)); appendFile(SD, "/dl1.txt", uSDstr);

appendFile(SD, "/dl1.txt", ";");
///Grava ATM, Gases organicos, umidade
intToChar(pres, uSDstr, sizeof(uSDstr)); appendFile(SD, "/dl1.txt",
```

 doi:10.20944/preprints202406.1329.v1


```
uSDstr);
appendFile(SD, “/dl1.txt”, “;”); intToChar(TVOC, uSDstr,
sizeof(uSDstr));
appendFile(SD, “/dl1.txt”, uSDstr); appendFile(SD,
“/dl1.txt”, “;”);
intToChar(humi, uSDstr, sizeof(uSDstr));

appendFile(SD, “/dl1.txt”, uSDstr); appendFile(SD, “/dl1.txt”,
“;”);
///Grava X Y Z
intToChar(acellX, uSDstr, sizeof(uSDstr));

appendFile(SD, “/dl1.txt”, uSDstr); appendFile(SD, “/dl1.txt”,
“;”);
intToChar(acellY, uSDstr, sizeof(uSDstr)); appendFile(SD,
“/dl1.txt”, uSDstr);
appendFile(SD, “/dl1.txt”, “;”);

intToChar(acellZ, uSDstr, sizeof(uSDstr)); appendFile(SD, “/dl1.txt”, uSDstr);

appendFile(SD, “/dl1.txt”, “;”); appendFile(SD,
“/dl1.txt”, “\n”);
}
}
void listDir(fs::FS &fs, const char * dirname, uint8_t levels) {
Serial.printf(“Listing directory: %s\n”, dirname);

File root = fs.open(dirname); if (!root) {
Serial.println(“Failed to open directory”);
return;
}
if (!root.isDirectory()) {
Serial.println(“Not a directory”);
return;
}

File file = root.openNextFile(); while (file) {

if (file.isDirectory()) { Serial.print(“      DIR : “);

Serial.println(file.name()); if (levels) {
listDir(fs, file.name(), levels -

1);
```

 doi:10.20944/preprints202406.1329.v1


```
}
} else {
Serial.print("     FILE: ");


Serial.print(file.name()); Serial.print("   SIZE: ");

Serial.println(file.size());
}
file = root.openNextFile();
}
}

void
createDir(fs::FS &fs, const char * path) { Serial.printf("Creating Dir: %s\n",
path);
if (fs.mkdir(path)) { Serial.println("Dir created");
} else {

Serial.println("mkdir failed");
}
}

void removeDir(fs::FS &fs, const char
* path) {
Serial.printf("Removing Dir: %s\n", path); if (fs.rmdir(path)) {

Serial.println("Dir removed");
} else {
Serial.println("rmdir failed");
}
}

void readFile(fs::FS &fs, const char * path) { Serial.printf("Reading file: %s\n", path);
File file = fs.open(path); if
(!file) {
Serial.println("Failed to open file for reading"); return;

}

Serial.print("Read from file: "); while (file.available()) {
```

 doi:10.20944/preprints202406.1329.v1


```
Serial.write(file.read());
}
file.close();
}

void writeFile(fs::FS &fs, const char * path, const char * message) {
Serial.printf("Writing file: %s\n", path);

File file = fs.open(path, FILE_WRITE); if (!file) {

Serial.println("Failed to open file for writing"); return;
}
if (file.print(message)) {
Serial.println("File written");
} else {

Serial.println("Write failed");
}
file.close();
}

void appendFile(fs::FS
&fs, const char * path, const char * message) { Serial.printf("Appending to file:
%s\n", path);

File file = fs.open(path, FILE_APPEND); if (!file) {

Serial.println("Failed to open file for appending"); return;
}
if (file.print(message)) {
Serial.println("Message appended");
} else {

Serial.println("Append failed");
}
file.close();
}

void renameFile(fs::FS
&fs, const char * path1, const char * path2) { Serial.printf("Renaming file %s to
%s\n", path1, path2);
if (fs.rename(path1, path2)) {
Serial.println("File renamed");
} else {
```

 doi:10.20944/preprints202406.1329.v1


```
Serial.println("Rename failed");
}
}

void
deleteFile(fs::FS &fs, const char * path) { Serial.printf("Deleting file:
%s\n", path);
if (fs.remove(path)) { Serial.println("File deleted");

} else {
Serial.println("Delete failed");
}
}

void testFileIO(fs::FS &fs, const char * path) {
File file = fs.open(path); static uint8_t buf[512];

size_t len = 0;
uint32_t start = millis(); uint32_t end = start;
if (file) { len
= file.size();
size_t flen = len; start = millis(); while (len) {
size_t toRead = len;
if (toRead > 512) { toRead = 512;
}

file.read(buf, toRead); len -= toRead;
}
end = millis() - start;

Serial.printf("%u bytes read for %u ms\n", flen, end); file.close();
} else
{
Serial.println("Failed to open file for reading");
}
file =
fs.open(path, FILE_WRITE); if (!file) {

Serial.println("Failed to open file for writing");
return;
}
size_t i;
start = millis();
for (i = 0; i < 2048; i++) {
```

 doi:10.20944/preprints202406.1329.v1


```
file.write(buf, 512);
}
end = millis() - start;
Serial.printf("%u
bytes written for %u ms\n", 2048 * 512, end); file.close();
}
```

References

1. Comisión Económica para América Latina y el Caribe. Índice ODS 2021 para América Latina y el Caribe.pdf. Bogotá, Colombia, ago. 2022.
2. Machado, N. Transporte marítimo adota ambição de emissões líquidas zero até 2050. epbr. Link: https://epbr.com.br/transporte-maritimo-adota-ambicao-de-emissoes-liquidas-zero-ate-2050/. Access in: 19 mai. 2024.
3. Mordor Intelligence. Tamanho do mercado de transporte de carga marítima e análise de participação – Tendências e previsões de crescimento (2024 – 2029). Link: https://www.mordorintelligence.com/pt/industry-reports/global-maritime-freight-transport-market. Acess in: 19 mai. 2024.
4. Sachs, J.; Kroll, C.; Lafortune, G.; Fuller, G.; Woelm, F. Sustainable Development Report 2022. 1. ed. Cambridge University Press, 2022. doi: 10.1017/9781009210058.
5. IEA; IRENA; UNSD; World Bank; WHO. Tracking SDG7: The Energy Progress Report 2022. Washington DC, 2022.
6. Rucks, S.; Leite, L.; Ferreira, I.; Salve, S.; Muniz, E. Relatório Anual 2022 ONU Brasil. Nações Unidas, Brasília, DF, mar. 2023.
7. Mccaffery, C.; Zhang, H.; Karavalakis, G.; Durbin, D. T.; Miller, J.; Johnson, C. K. Sources of air pollutants from a Tier 2 ocean-going container vessel: main engine, auxiliary boiler. Atmospheric Environment, 2021. ISSN 1352-2310.
8. Seungman, H.; Byongug, J.; Jang, H.; Park, C.; Ku, B. A framework for determining the life cycle GHG emissions of fossil marine fuels in countries reliant on imported energy through maritime transportation: A case study of South Korea. Science of the Total Environment, v. 123, n. 4, p. 567-580, 2023. ISSN 0048-9697.
9. Salgado, D. A propulsão elétrica nos navios mercantes. Trabalho de conclusão de curso. Marinha do Brasil, Centro de Instrução Almirante Graça Aranha, Curso de Oficiais da Marinha Mercante, 2013. Disponível em: https://www.redebim.dphdm.mar.mil.br/vinculos/000005/0000054d.pdf. Acesso em: 19 mai. 2024.
10. Silva, D. Motores à combustão interna. MOTUL. Disponível em: https://motulexpert.com.br/fenomenos-de-falhas-em-motor-a-combustao-interna/. Access in: 19 mai. 2024.