Projeto aplicado - Estação Meteorológica

Versão 3.0

Aluno: Marcelo Miguel Cardoso
Matrícula: 19102371

07/12/2022

Sumário

Lista de Figuras

Lista de Tabelas

1 Links para acesso

Github do projeto

2 Observações do texto

Algumas adaptações do texto foram necessárias para melhor legibilidade e organização:

1. Os códigos fontes aqui citados não estão completos, omite-se partes que podem poluir muito a legibilidade, sendo assim os programas podem ser encontrados na integra no Github;

2. Este texto consiste da terceira versão do relatório onde algumas alterações foram feitas, acrecidas e serão mencionadas ao longo do texto; Foi adicionada uma descrição do funcionamento do sistema.

3. Cada versão mantém a versão antiga do texto mas acrescenta atualizações, sendo assim não é necessária consulta nas versões anteriores;

3 Introdução

3.1 Definição do projeto

O projeto proposto será implementar uma estação meteorológica que através da aplicação de *machine learning* que possibilite prever chuva, utilizando para isso programação orientada a objeto. Mais especificamente *machine learning* embarcada, programada em linguagem C++. Com isto será desenvolvida a habilidade de desenvolver software e hardware conjuntamente aplicando os conceitos vistos na disciplina.

3.2 Objetivos

Os objetivos a seguir foram obtidos da proposta e já foram adaptados com base nas decisões do sistema que será descrita a seguir.

O projeto proposto visa aplicar os conceitos aprendidos na disciplina:

- Linguagem C++, com Herança, Friends, Template, Funções virtuais, Tratamento de exceções, Sobrecarga de operadores, Polimorfismo (com classes abstratas);

- Entender os desafios do projeto integrado de software/hardware para sistemas embarcados;

- Praticar o fluxo completo de projeto de sistemas em C++ para sistemas embarcados.

- Utilizar os conceitos de programação orientada a objetos, principalmente o uso de polimorfismo com classes abstratas visando a portabilidade do código (quando necessário).

Além disso são dadas algumas demandas que devem ser cumpridas:

- Minimizar consumo energético: utilizar um sistema que seja voltado a menor consumo e IoT, e recursos de minimização de consumo tais como modos *sleep*;
- Possibilitar a autonomia energética: reduzindo o consumo poderá ser alimentado por bateria e deverá ter sistema de recarga com bateria de backup ou consumo somente sob demanda;
- Aprender com sensores as características do ambiente ao qual o sistema será inserido e com base nisto prever se haverá probabilidade de chuva ou não: será usado o sistema apresentado na disciplina para *machine learning* embarcada: **TensorFlow Lite**;
- Se houver erro na previsão deverá enviar ao servidor e com isso melhorar o sistema: será aplicado um sistema de envio para rede via *bluetooth* ou *Wi-fi*;
- *Log* de operações utilizando uma estrutura de fila para armazenar as informações sobre os eventos ocorridos. Registrar temperatura, umidade, luminosidade, acertos e erros de previsão, com troca de parâmetros com o servidor além disso pretende-se registrar a data, hora, ID do sistema: possivelmente será usado um cartão de memória para isto;
- Para possibilitar a consulta das informações, o *log* precisará ser transferido para um computador hospedeiro (desktop ou notebook). Durante o envio do *log* para o hospedeiro, a fila deverá ser esvaziada (o *log* será reinicializado), e os dados transferidos para outra estrutura de dados no hospedeiro (uma lista, por exemplo). O software do hospedeiro também deve ser escrito em C++;
- A comunicação com o hospedeiro deve ser realizada por intermédio de uma UART (pode ser utilizado um cabo de conversão UART/USB). Além disso, deve ser previsto algum meio de comunicação *wireless* para transferência de dados para um *smartphone*.

3.2.1 Software hospedeiro

Usuário *admim* pode consultar:

- Listar todos os eventos ocorridos em um determinado intervalo de datas;
- Obter os dados adquiridos, enviados ou não, em um determinado intervalo de datas.

- O software do hospedeiro deve ser escrito em C++, sendo que no *smartphone* poderão ser utilizados dados sintéticos para representar as informações obtidas do controlador proposto. Não é necessário realizar a transferência de dados do sistema embarcado para o *smartphone*, mas é preciso descrever o hardware que seria necessário para realizar a transferência wireless (ex. módulo bluetooth, wifi) do sistema embarcado para o *smartphone*.

- Para o caso de um computador (ex. Notebook, desktop), o software deve ser escrito visando a recepção dos dados via porta serial (com cabo).

3.3 Ferramentas utilizadas

3.3.1 Git/Github

Git/Github para o controle de versão e compartilhamento dos programas com o professor: códigos comentados.

Git e Github são utilizados no dia a dia das pessoas que criam software por um motivo bem simples: ter uma forma fácil de gerenciar o código fonte da aplicação.

3.3.2 Overleaf

Para elaboração do texto em LaTeX, o site é gratuito e conta com muitas bibliotecas e documentações necessárias para redigir textos com diferentes recursos tais como facilidade de incluir códigos fonte de maneira adequada.

3.3.3 Arduino IDE

Para desenvolvimento do software embarcado será usada a IDE Arduino que é gratuita e conta com suporte da comunidade. Esta IDE pode ser utilizada para programar diferentes famílias de plcas tasi como micrcontroladores AVR, ARM (ex.: STM32) e a placa utilizada no projeto: ESP32.

Há muita documentação na internet sobre sua utilização e além disso muitos exemplos podem ser encontrados de maneira livre, distribuida or intusiastas da eletronica.

3.3.4 Compilador G++ Linux

A GNU Compiler Collection, comumente conhecida como GCC, é um conjunto de compiladores e ferramentas de desenvolvimento disponíveis para Linux, Windows, vários BSDs e uma ampla variedade de outros sistemas operacionais.

Este compilador será usado para compilar os softwares do computador e permitirá criar o executável para rodar em Linux.

3.3.5 Sublime Text

Sublime Text é um editor de código fonte de plataforma cruzada shareware. Ele suporta nativamente muitas linguagens de programação e linguagens de marcação. Os

usuários podem expandir sua funcionalidade com plug-ins, normalmente construídos pela comunidade e mantidos sob licenças de software livre. Para facilitar os plugins.

3.3.6 Conversor FTDI UART USB

Para comunicação UART com computador será usado um adaptador que permite a conversão simplificada dos protocolos usados em microcontroladores para USB comumente encontrada em computadores. Este adaptador é suportado nativamente pelo Windows e Linux e possibilita as principais taxas de comunicação.

3.3.7 Fonte de bancada

Agora viu-se a necessidade de utilizar uma fonte de bancada para avaliar o consumo total e assim estimar a duração da bateria;

3.3.8 Outras ferramentas

Multímetro e outras ferramentas de laboratório; Protoboard para montagem do circuito; Jumpers para montagem;

4 Hardware

Sabendo dos limites de consumo e dos objetivos necessários ao sistema. Pelas especificações sabemos que deverá ter pelo menos os seguintes sensores:

Sensor	**Localização**	**Modelo do sensor**
Temperatura	Interno	AHT21
	Externo	DHT11
Umidade	Interno	AHT21
	Externo	DHT11
Luminosidade	Interno	LDR 5mm
	Externo (Leste)	LDR 5mm
	Externo (Oeste)	LDR 5mm
	Externo (Sul)	LDR 5mm

Tabela 1: Tabela de sensores utilizados

4.1 Placa ESP-WROOM-32

Figura 1: ESP32

Processador principal: Microprocessador Tensilica Xtensa 32-bit LX6; Núcleo: 2 ou 1 (depende da variação); Todos os chips na série ESP32 são dual-core, com exceção do modelo ESP32-S0WD, que é single-core; Frequência de Clock: até 240 MHz; Performance: até 600 DMIPS; Processador secundário: Ultra low power; Conectividade: Wi-Fi: 802.11 b/g/n/e/i (802.11n @ 2.4 GHz até 150 Mbit/s); Bluetooth: v4.2 BR/EDR e Bluetooth Low Energy (BLE); ROM: 448 KiB – Usada em Boot e funções principais do ESP32; SRAM: 520 KiB – Usada para dados e instruções (programas); RTC slow SRAM: 8 KiB – Para acesso do co-processador em modo deep-sleep; RTC fast SRAM: 8 KiB – Para armazenamento de dados e uso de CPU em boot de RTC (relógio de tempo-real) do modo deep-sleep; eFuse: 1 Kbit – Dos quais 256 bits são usados para sistema (endereço MAC e configurações do chip), e os restantes 768 bits são reservados para aplicações incluindo criptografia da Flash e Chip-ID. Flash externa 4 MB; Periféricos: Periféricos: comunicação com suporte a DMA, 10 GPIOs com suporte a toque capacitivo, 16 canais de conversor SAR ADCs (conversor analógico-digital) de 12-bits, 2 canais de 8 bits DACs (conversor digital-analógico), 2 Interfaces I^2C (Inter-Integrated Circuit), 2 interfaces UART (universal asynchronous receiver/transmitter), Controlador CAN 2.0 (Controller Area Network),4 interfaces SPI (Serial Peripheral Interface), 2 interfaces I^2S (Integrated Inter-IC Sound), 16 canais de PWM (modulação por largura de pulso).

Este microcontrolador foi escolhido por possuir os recursos que serão usados no projeto, tais como protocolo de comunicação, pinos digitais, pinos analógicos, comunicação *wi-fi* entre outros que serão possivelmente usados. Além disso o ESP32 conta com modos de baixo consumo e RTC interno que serão usados.

Outro fator que foi levado em conta para escolha deste microcontrolador é sua facilidade de programação usando a plataforma Arduino, onde muitas bibliotecas, documentações, tutoriais podem ser encontrados. Sabe-se que profissionalmente a plataforma pode ser menos robusta em termos de software e sua estabilidade.

4.2 Sensor DHT11 umidade e temperatura

Figura 2: DHT11

Características: Elevada estabilidade,Baixo consumo, Resposta rápida, Saída digital via pino único; Faixa umidade: 20% a 90% RH, ±5.0%RH; Faixa temperatura: 0 a +50 °C, ±2.0 °C; Elemento sensor de temperatura: termistor 100K + / -1%; Elemento sensor de umidade: HR202; Tensão de alimentação: 3 a 5,5V DC; Corrente: 0.2 a 0.5 mA. Stand by: 100 a 150 uA; Dimensões: 15,5 x 12 x 5,5mm.

Este sensor foi escolhido pelas características de medição de temperatura e umidade que atendem as necessidades de precisão, além de ser de fácil programação utilizando bibliotecas já desenvolvidas que serão referenciadas posteriormente. Este modelo será usado no exterior por contar com encapsulamento mais robusto a intempéries.

Este modelo utilizará a biblioteca disponível para Arduino em ESP32.

4.3 Sensor AHT21 umidade e temperatura

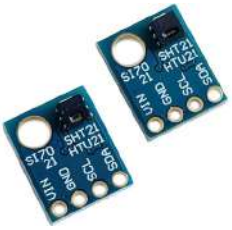

Figura 3: AHT21

Características: Tensão de alimentação: 2,2 a 5,5V; Comunicação: I2C; Precisão de umidade: ±3 %RH; intervalo de umidade: 0 a 100%RH; Precisão de temperatura: ±0,5°C; intervalo de temperatura: -40 a 80°C; Dimensões: 12cm x 12cm.

Este sensor foi escolhido pelas características de medição de temperatura e umidade que atendem as necessidades de precisão, além de ser de fácil programação utilizando bibliotecas já desenvolvidas que serão referenciadas posteriormente. Este modelo será usado no interior por ser mais preciso do que o anterior tanto em umidade como temperatura. Mas este modelo é menos favorável em ambientes externos.

Este modelo utilizará a biblioteca disponível para Arduino em ESP32.

4.4 Sensor LDR

Figura 4: LDR

Características: Diâmetro: 5mm; Tensão de operação (máxima): 150VDC; Potência máxima: 100mW; Espectro: 540nm; Temperatura de operação: -30° a 70° C; Resistência (presença de luz): 1MO; Resistência (ausência de luz): 10 a 20KO.

O resistor dependente de luz (LDR) será usado para medir a luz em unidades de *lux* e assim poder comparar ao modelo que será usado. Para utiliza-lo será necessário apenas a leitura analógica de um pino que será ligado como divisor de tensão. A quantidade de *lux* será convertida e explicada no programa.

4.5 Fonte 5V

Entrada 110V/220V; Saída 5V; Modelo: Hi-Link PM01; Potência: 3W; Corrente de saída máxima para um longo período de tempo: >s600mA; Corrente de saída máxima de curto período de tempo: >1000mA.

Figura 5: Fonte 5V

Esta fonte será usada para carga da bateria em momentos em que o sistema esteja ligado a rede.

4.6 Módulo TP4056

CI controlador: TP4056 (Datasheet);Tensão de operação: 5V; Capacidade máxima de carga: 1A (ajustável);Tensão de corte na saída: 4.2V +/- 1%; Proteção contra sobrecarga; Conexão micro USB; Temperatura de operação: -10^{o}C à 85^{o}C; Dimensões: 26 x 17 x 5mm.

Figura 6: TP4056

Este controlador é necessário para que a carga da bateria seja feita de maneira correta, usando para isso controle por corrente constante e após tensão constante. Assim, a bateria é protegida e mantém sua vida útil por mais tempo. O controlador também desliga a saída em caso de baixa carga da bateria.

4.7 Bateria de Li-ion Recarregável

Modelo: NCR 18650B (Datasheet); Tensão nominal: 3,7 V; Capacidade: 3400 mAh; Máxima corrente de carga: 2A; Corrente de descarga: 1 a 2,9 C; Tamanho: 18 mm x 69,0 mm; Peso: 48 g.

Figura 7: LDR

Esta bateria foi escolhida por ter as características mínimas e por já possuí-la. As especificações são compatíveis com o controlador de carga e fonte e fornecerá algumas horas de autonomia para o sistema de maneira plena e até mesmo dias se for usado modos de baixíssimo consumo.

4.8 Módulo Cartão SD

Tensão de operação: 3,3 ou 5V; Interface SPI: MOSI, SCK, MISO e CS; Dimensões: 41 x 24mm;

Figura 8: SD

O módulo de cartão SD em conjunto com um cartão SD poderá ser usado como *datalogger* do sistema.

Para utiliza-lo será necessário a biblioteca para cartão SD em conjunto com a biblioteca SPI disponível na plataforma Arduino.

4.9 Display Oled

Tamanho 0.96"; Resolução: 128 x 64; Tensão 3.3 a 5V DC; Potência: 0.06W Max; Interface: I2C

Figura 9: Display oled

5 Hardware integração

O projeto exige que haja conectividade entre diferentes meios e o sistema embarcado, sendo assim necessitará de dispositivos para tal.

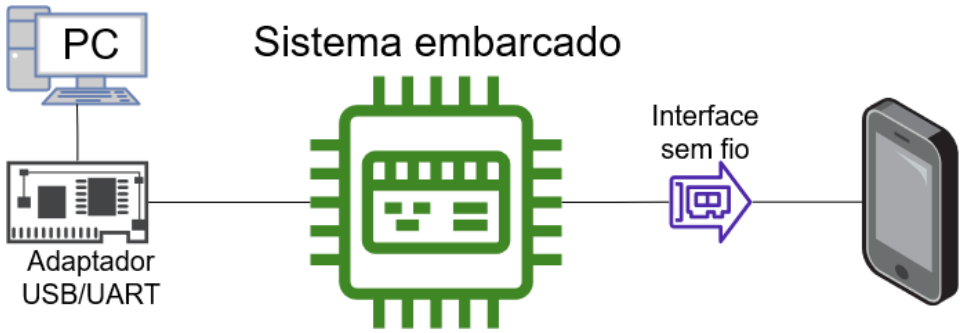

Figura 10: Diagrama de interfaces (fonte: Autoria própria)

5.1 Conectividade com computador

Para comunicação com computador será usado um adaptador e a conectividade UART disponível na placa ESP32.

O adaptador FTDI utiliza tensões de 3V3 ou 5V que são adequadas a placa utilizada.

Figura 11: Adaptador FTDI UART/USB

5.2 Conectividade com smartphone

Na primeira versão imaginava-se utilizar o seguinte adaptador para interfacear comunicação UART via Bluetooth: O módulo apresentado a seguir será usado para transmitir ao smartphone. Este pode ser facilmente programado pois utilizará também UART que está disponível na biblioteca Arduino para ESP32.

A conectividade com smartphone será feita através de bluetooth utilizando para isto um módulo que usa UART e o transmite ao smartphone conectado.

Figura 12: Módulo HC05

Mas na segunda versão do texto optou-se por utilizar o Bluetooth de baixo consumo (BLE) interno ao ESP32 visto que reduzirá a complexidade bem como liberará 2 pinos de comunicação serial que poderiam ser úteis em eventuais *upgrades*. Dessa forma o esquemático foi atualizado e será apresentado a seguir.

5.3 Módulo RTC

Foi incluído um módulo RTC na verção 2. O módulo DS1307 conta com uma bateria e serve para registrar a data e hora exatas inclusive em situação de falta de energia/-bateria. Para isso ele possui uma pilha exclusiva para manter as informação e o clock atualizado.

O protocolo que este módulo usa é I2C e portanto será ligado no mesmo barramento que o oled e sensor interno.

Figura 13: Módulo RTC DS1307

5.4 Esquemático do sistema embarcado

Esquemático versão 1: A seguir é apresentado como ocorrerá a montagem da placa com seus periféricos, sensores, display e sistema de energia.

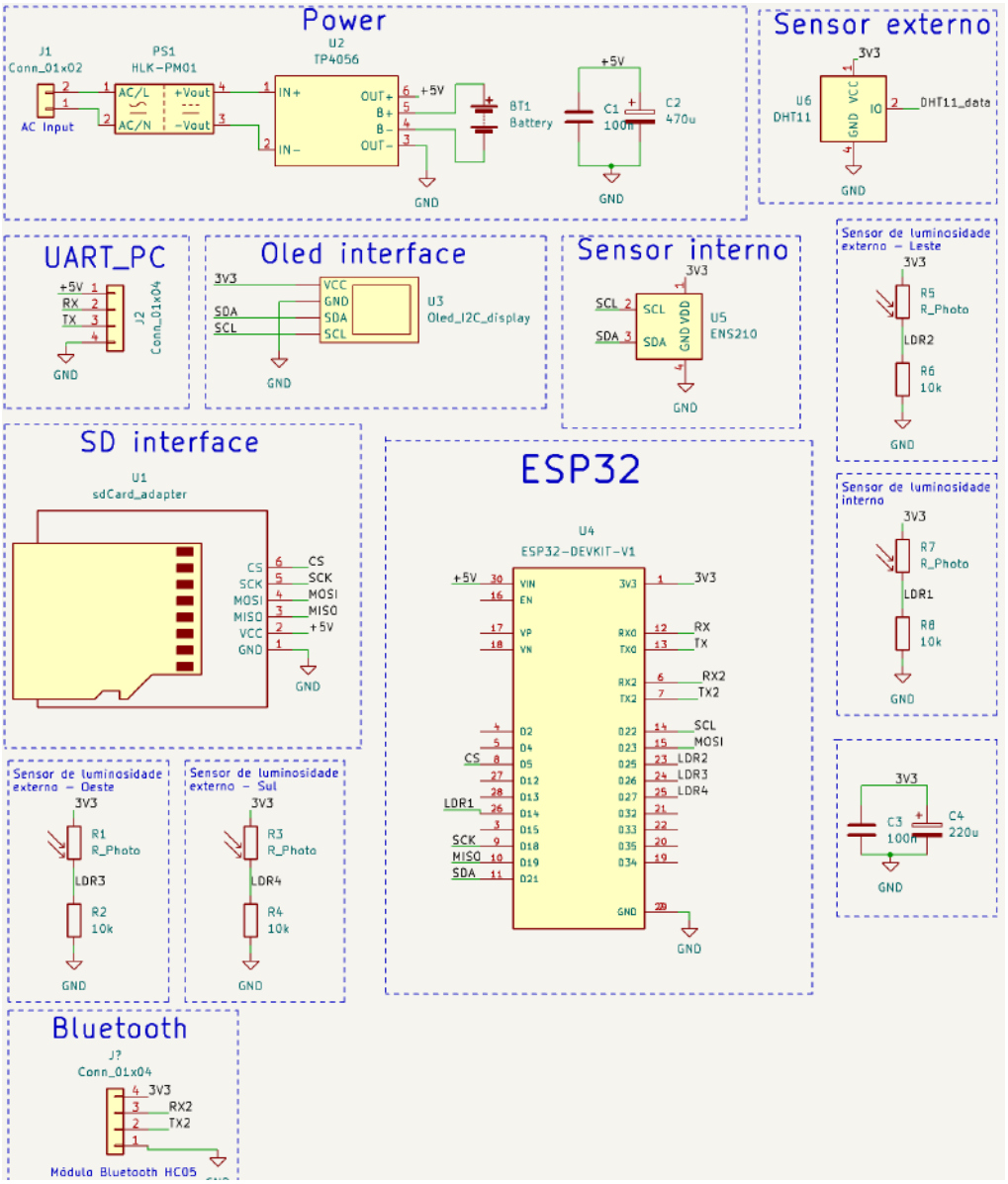

Figura 14: Esquemático V1 - hardware (Fonte: Autoria própria)

A seguir as atualizações do esquemático: Removeu-se o módulo bluetooth HC05 e trocou-se os pinos serial para interface com PC. Além disso o CS do módulo SD foi alterado para melhor facilidade na montagem.

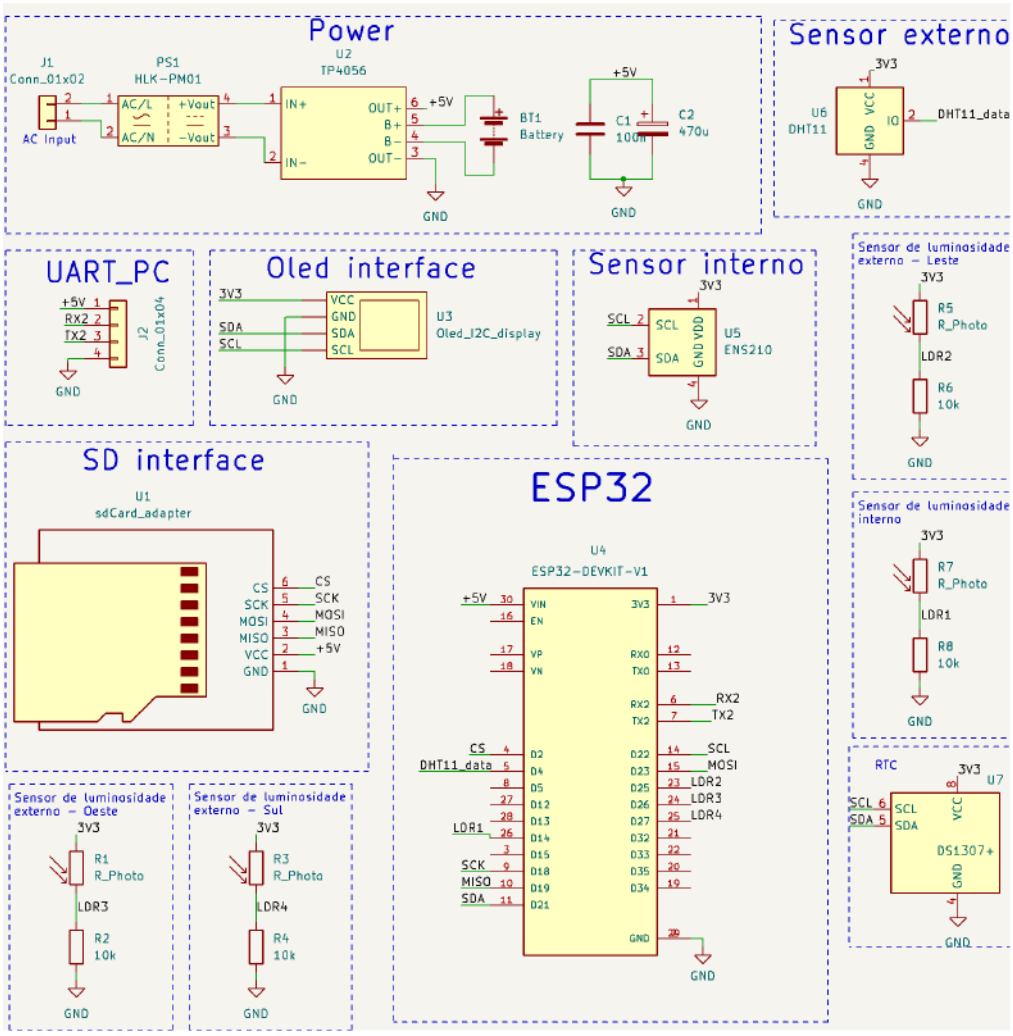

Figura 15: Esquemático V2 - hardware (Fonte: Autoria própria)

6 Montagem do sistema

A versão 2 do texto traz a primeira montagem do sistema completo:

Na figura abaixo podemos notar os sensores LDR na parte superior direita, estes estão apontando para leste, oeste e sul. Abaixo o sensor LDR interno e o display oled. Há ainda o sensor DHT11 de temperatura e umidade na parte superior. Módulo SD na parte direita inferior e por fim o sensor de umidade e temperatura interna na parte inferior esquerda.

Nota-se que ainda a parte de energia não foi incluída pois está será deixada para o final do projeto visando simplicidade do sistema para programa-lo.

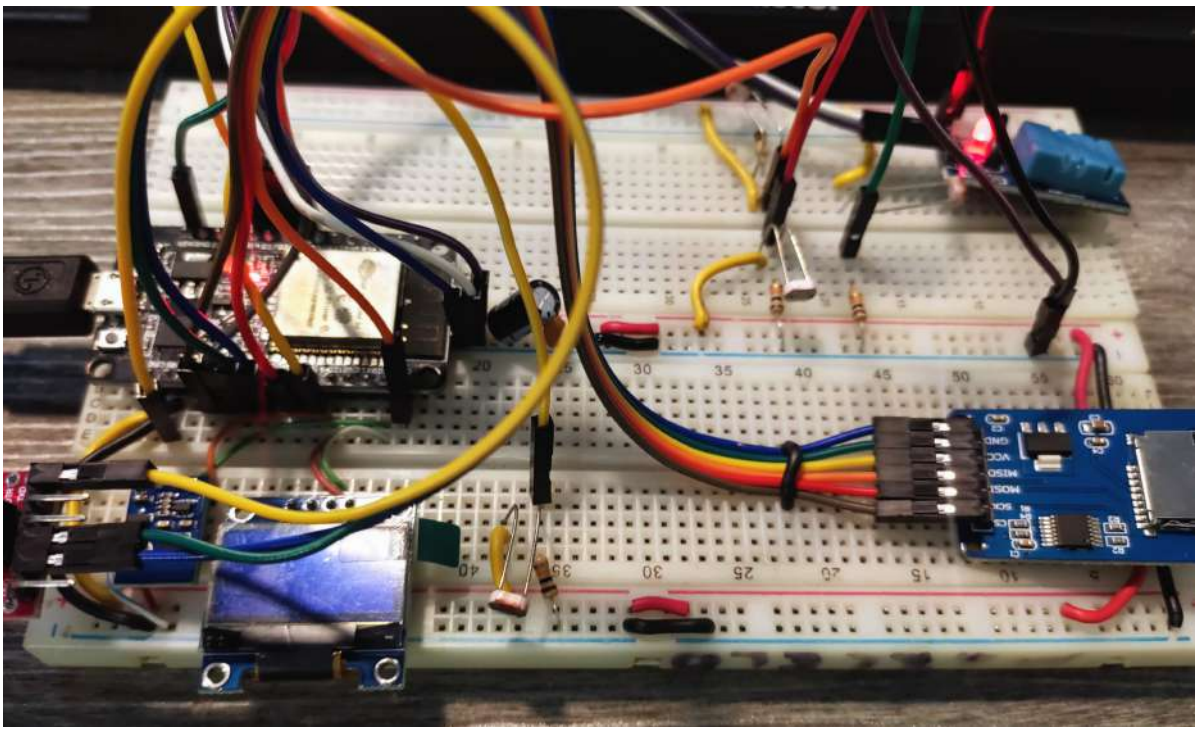

Figura 16: Montagem 1 do sistema completo

7 Funcionamento esperado do sistema

A idea inicial é obter os dados dos sensores, salvar no cartão de memoria a cada intervalo de tempo programado (exemplo a cada minuto). Assim, para o usuário interagir com o sistema, será feita uma interface com menus para consulta de dados atuais e histórico, bem como exibir a previsão do tempo se for finalizado a tempo.

Para tornar mais genérico, o sistema rodaria o terminal com menus no hardware embarcado, exibindo as mensagens no terminal tanto bluetooth como no computador via USB ambos melhor descritos ao longo do texto.

Ainda não se obteve sucesso no desenvolvimento dos menus, mas algumas funções foram adicionadas:

- Timer para salvar log a cada 5 segundos por exemplo;
- Função membro para contagem de linhas já salvas no cartão SD;
- Função para ler somente uma linha específica;
- Função para consultar uma data (em desenvolvimento);
- Menus (em desenvolvimento);

8 Software embarcado

8.1 Diagrama de classes - Sistema embarcado

Versão 1 foi fornecido o diagrama de classes preliminar abaixo

Abaixo há o diagrama preliminar de classes que será embarcado no ESP32. O programa terá classes para sensores, interface serial, interface display e bluetooth.

Além disso possivelmente haverá menus para acesso aos recursos tais como consulta do log, mudança de exibição no display, envio via bluetooth etc.

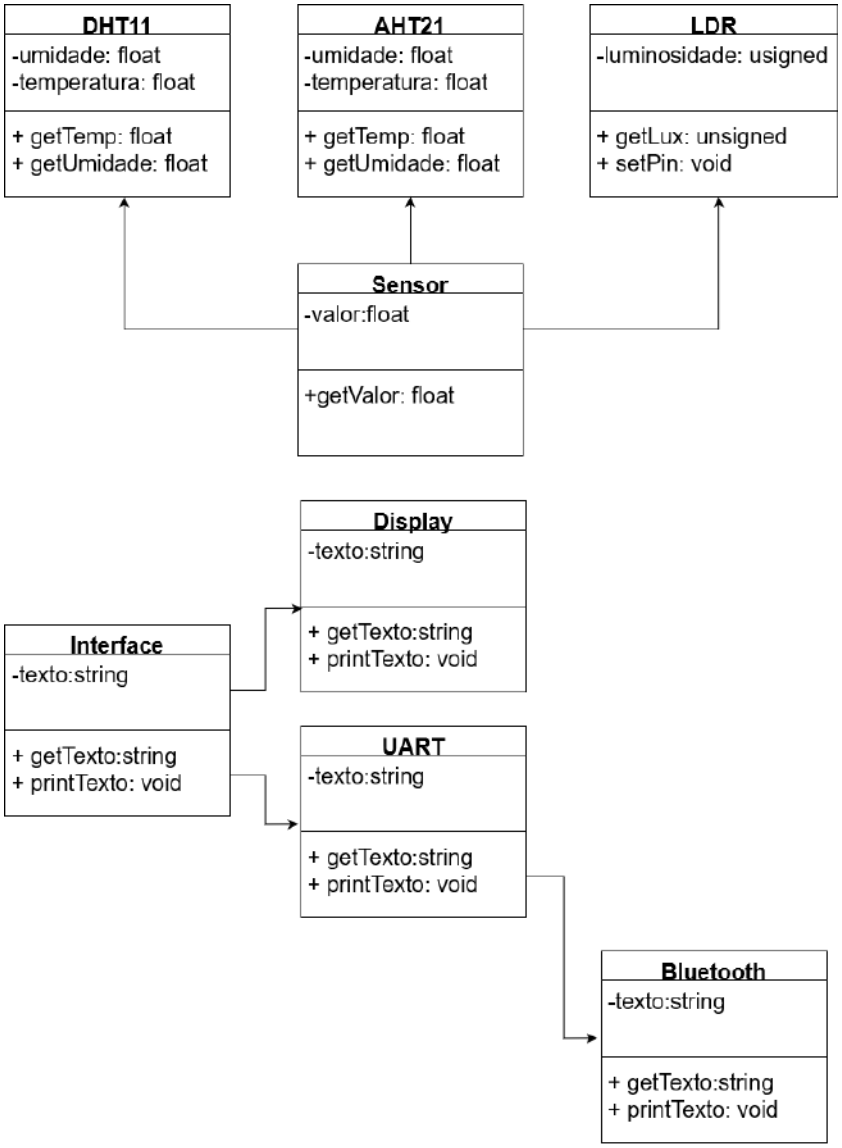

Figura 17: Diagrama de classes do software embarcado (preliminar)

Pretende-se aprender a utilizar o relógio de tempo real com bateria de backup (modo de baixo consumo) e assim será necessárias mais classes, métodos e possivelmente algum

dispositivo físico a mais.

Diagrama de classes versão 2: Agora temos a versão atualizada do diagrama de classes apresentado abaixo. Nele separou-se os sensores, RTC, cartão SD, sensor de luminosidade. Ainda pretende-se implementar o display oled mas este será deixado para o final pois é de menor prioridade.

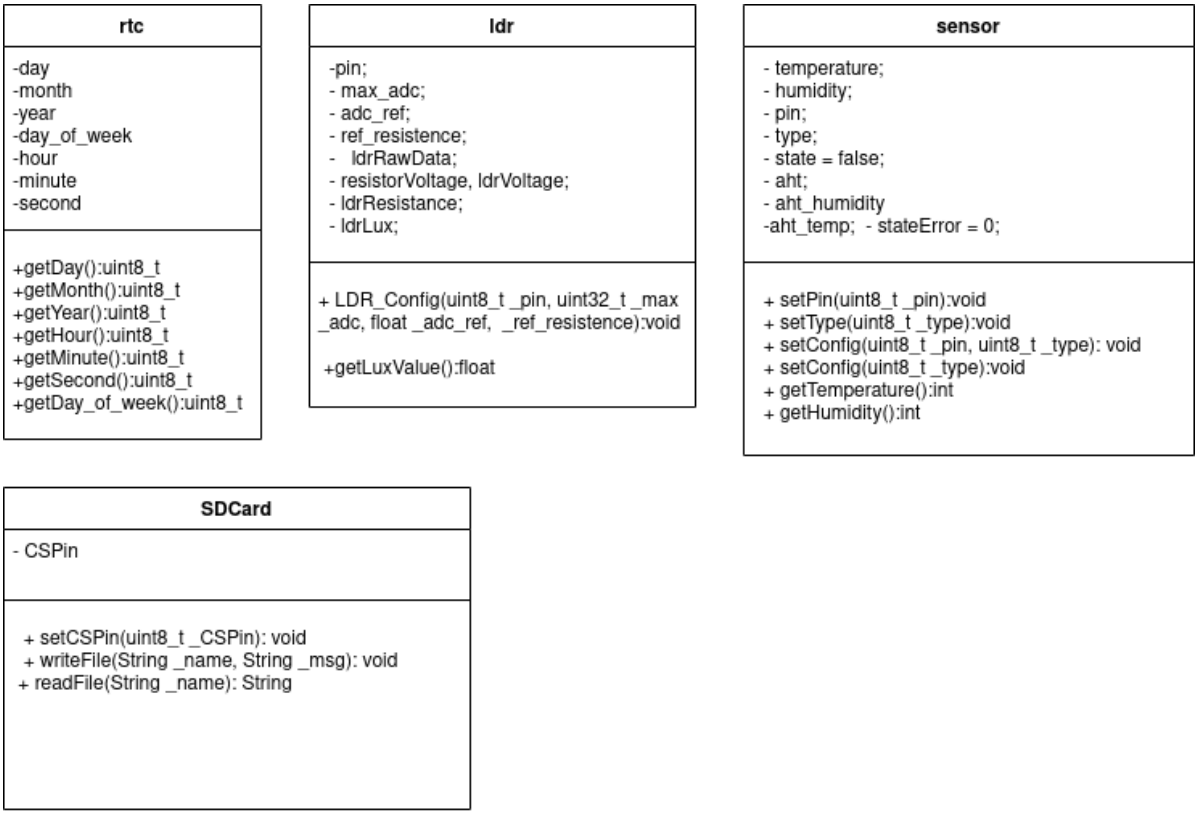

Figura 18: Diagrama de classes do software embarcado Versão 2

Diagrama de arquivos versão 3:

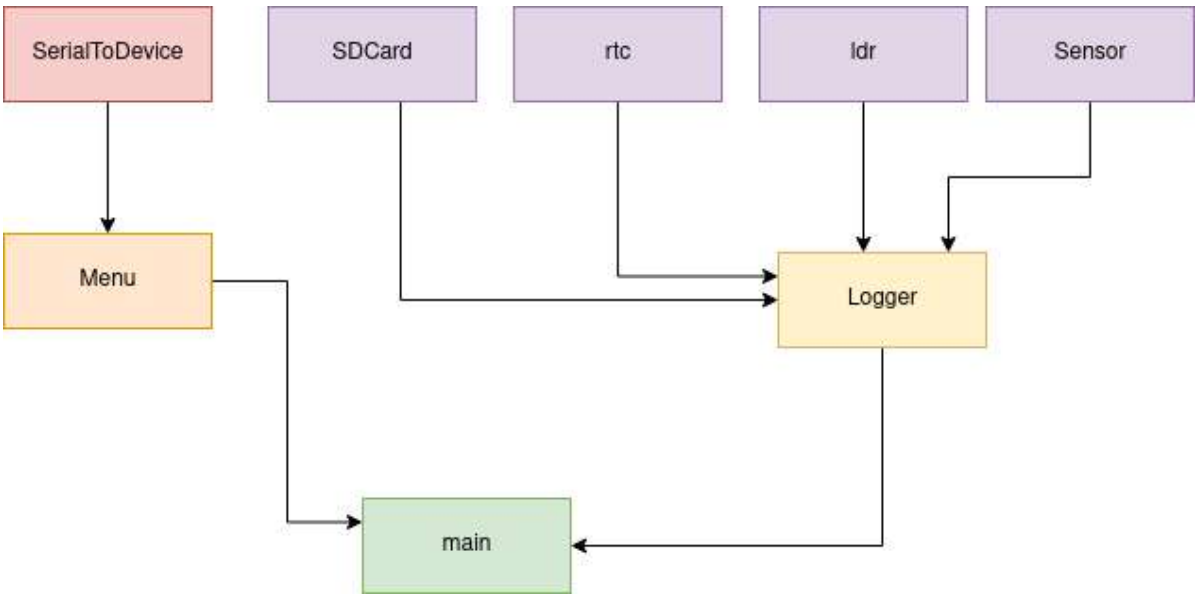

Figura 19: Diagrama de arquivos hpp versão 3

8.2 Testes dos dispositivos

A partir de agora serão feitos testes com cada dispositivo individualmente para validar as conexões com o ESP32

8.2.1 Teste do display Oled

Primeiramente como já mencionado, o display oled e o sensor interno utilizam protocolo I2C que portanto poderá ser testado ao realizar um sistemas *scan* nos 128 endereços.

```
#include <Wire.h> //biblioteca Arduino para I2C

void setup()
{
  Wire.begin();

  Serial.begin(9600);
  while (!Serial);          // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;
```

```
  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println(" !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);          // wait 5 seconds for next scan
}
```

Dessa forma obtemos a seguinte saída:

```
Scanning...
I2C device found at address 0x38 !
I2C device found at address 0x3C !
done
```

No github pode ser encontrado um código de teste do oled, o qual foi realizado com sucesso.

8.2.2 Teste do sensor interno

Como já mencionado anteriormente, se o endereço foi encontrado o sensor está funcionando perfeitamente. Mas para confirmar podemos fazer *upload* de um exemplo para leitura de umidade e temperatura. Este exemplo pode ser encontrado na biblioteca do sensor. Ao realizar o teste obtemos a saída esperada:

```
    Temperature 28.21 deg C
    Humidity: 62.70 % rH
```

8.2.3 Teste do sensor externo

O código fornecido pela biblioteca foi testado e obtemos a seguinte saída:

```
Temperature: 27.90C
Humidity: 72.00%
```

8.2.4 Testes dos sensores de luminosidade

O seguinte software foi usado para medir luz em *lux* através do LDR ligado ao conversor AD interno do ESP32:

```
#define LDR_PIN                 14
#define MAX_ADC_READING         4095
#define ADC_REF_VOLTAGE         3.3
#define REF_RESISTANCE          10000 // measure this for best results
#define LUX_CALC_SCALAR         12518931
#define LUX_CALC_EXPONENT       -1.405


void setup(void)
{
  Serial.begin(9600);
  Serial.println(F("Light Sensor Test")); Serial.println("");


}

void loop(void)
{

   int   ldrRawData;
   float resistorVoltage, ldrVoltage;
   float ldrResistance;
   float ldrLux;

  // Perform the analog to digital conversion
```

```
  ldrRawData = analogRead(LDR_PIN);

  // RESISTOR VOLTAGE_CONVERSION
  // Convert the raw digital data back to the voltage that was measured on the
       analog pin
  resistorVoltage = (float)ldrRawData / MAX_ADC_READING * ADC_REF_VOLTAGE;

  // voltage across the LDR is the 5V supply minus the 5k resistor voltage
  ldrVoltage = ADC_REF_VOLTAGE - resistorVoltage;

  // LDR_RESISTANCE_CONVERSION
  // resistance that the LDR would have for that voltage
  ldrResistance = ldrVoltage/resistorVoltage * REF_RESISTANCE;

  // LDR_LUX
  // Change the code below to the proper conversion from ldrResistance to
  // ldrLux
  ldrLux = LUX_CALC_SCALAR * pow(ldrResistance, LUX_CALC_EXPONENT);

  // print out the results
  Serial.print("LDR Raw Data : "); Serial.println(ldrRawData);
  Serial.print("LDR Voltage : "); Serial.print(ldrVoltage); Serial.println("
      volts");
  Serial.print("LDR Resistance : "); Serial.print(ldrResistance); Serial.
      println(" Ohms");
  Serial.print("LDR Illuminance: "); Serial.print(ldrLux); Serial.println("
      lux");

  delay(250);
}
```

Baseado na seguinte referencia Medir Lux com LDR

8.2.5 Teste do módulo SD

O código armazenado no Github foi testado e obteve-se sucesso. Cabe salientar que o módulo SD utiliza protocolo de comunicação SPI que necessita da biblioteca SPI.h do Arduino.

8.2.6 DS1307

O módulo RTC utilizado usa protocolo I2C e foi incluido para fornecer a data e hora exata sem que em caso de falta de energia e falta de batria ainda haja a bateria de backup responsável por manter a data e hora correndo.

O seguinte código é responsável por ajustar a data e hora atuais.

```
// Date and time functions using a DS1307 RTC connected via I2C and Wire lib
```

```
#include <Wire.h>
#include "RTClib.h"

RTC_DS1307 rtc;

char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "
    Thursday", "Friday", "Saturday"};

void setup () {
  Serial.begin(9600);
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (! rtc.isrunning()) {
    Serial.println("RTC is NOT running!");
    // following line sets the RTC to the date & time this sketch was compiled
    // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date & time, for example to set
    // January 21, 2014 at 3am you would call:
    rtc.adjust(DateTime(2022, 11, 26, 19, 45, 0));
  }
}

void loop () {
    DateTime now = rtc.now();
    Serial.print(now.day(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.year(), DEC);
    Serial.print(" (");
    Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
    Serial.print(") ");
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();

    delay(3000); //Print date and time every 3 sec
}
```

8.3 Código principal - embarcado

```
//inclusão das bibliotecas utilizadas:
#include "Sensor.hpp"
#include "ldr.hpp"
#include "rtc.hpp"
#include "SDCard.hpp"

//declaração dos objetos
SDCard datalog;
Sensor TH_interno;
Sensor TH_externo;
ldr LDR_sul;
ldr LDR_leste;
ldr LDR_oeste;
ldr LDR_interno;
rtc relogio;

//variavel auxiliar para somar strings de maniera mais visivel
String registro , data_registro, horario_registro;

//configuração iniciais

void setup() {

  Serial.begin(115200); //UART para debugar
  //configuração dos sensores e SD

  datalog.setCSPin(2);
  TH_externo.setConfig(4,dht_11);
  TH_interno.setConfig(aht_20);
  LDR_sul.LDR_Config(27, 4096, 3.3, 10000);
  LDR_oeste.LDR_Config(26, 4096, 3.3, 10000);
  LDR_leste.LDR_Config(25, 4096, 3.3, 10000);
  LDR_interno.LDR_Config(14, 4096, 3.3, 10000);

  //Cabeçalho rodar somente uma vez
  //datalog.writeFile("/datalog.txt","Data,Horario,Temp_int,Hum_int,Temp_ext,
      Hum_ext,Lux_int,Lux_oest,Lux_leste,Lux_sul\r\n");

}

void loop() {
```

```
  //a cada 5 segundos le todos os sensores e grava no cartão SD em arquivo de
      texto separado por ','

  delay(5000);
  data_registro = (String)relogio.getDay() + "/" + (String)relogio.getMonth()
      + "/" + (String)relogio.getYear();
  horario_registro = (String)relogio.getHour() + ":" + (String)relogio.
      getMinute() + ":" + relogio.getSecond();

  Serial.println(registro);

  registro = (String)data_registro + "," + (String)horario_registro + "," + (
      String)TH_interno.getTemperature() + ","
  + (String)TH_interno.getHumidity() + "," + (String)TH_externo.getTemperature
      () + ","
  + (String)TH_externo.getHumidity() + "," + (String)LDR_interno.getLuxValue()
       + "," + (String)LDR_oeste.getLuxValue() + ","
  + (String)LDR_leste.getLuxValue() + "," + (String)LDR_sul.getLuxValue() + "\
      r\n";

  datalog.writeFile("datalog.csv", registro);
}
```

8.4 Código embarcado resultados

Com o código anteriormente apresentado podemos gravar no cartão SD eventos a cada período (neste caso optou-se por 5 segundos). Abaixo obtemos uma amostra dos resultados:

A	B	C	D	E	F	G	H	I	J
30/11/230	15:18:54	26	86	26	86	1.35	0.69	8.48	8.91
30/11/230	15:19:0	26	86	26	86	1.64	0.76	8.62	9.13
30/11/230	15:19:5	26	36	65	36	1.71	0.84	9.57	9.75
30/11/230	15:19:10	26	85	26	85	1.74	0.71	9.84	9.56
30/11/230	15:19:15	26	84	26	84	1.81	0.74	10.21	9.27
30/11/230	15:19:21	26	85	26	85	1.85	0.69	9.97	8.44
30/11/230	15:19:26	26	85	26	85	1.51	0.49	7.37	6.64
30/11/230	15:19:31	26	85	26	85	1.67	0.61	8.40	7.34
30/11/230	15:19:36	26	85	26	85	1.57	0.52	7.32	7.27

Figura 20: Dados obtidos via cartão SD

Comparou-se a temperatura ambiente com um multímetro e os resultados estão satisfatórios. Temos apresentados a data, hora, temperatura interna, umidade interna, temperatura externa, umidade externa, luminosidade interna, luminosidade oeste leste e sul respectivamente. Neste teste os sensores não foram dispostos no ambiente externo, isso explica a proximidade dos valores.

8.5 Código do sistema embarcado - próximas atualizações

Nos próximos dias serão implementadas as funções de comunicação entre PC e smartphone de modo que seja possível a consulta do *log* armazenado no cartão SD.

Na versão 3 foi adicionada a busca por linha no *log* para comparar com o valor solicitado pelo usuário.

9 Software PC

Primeiramente devemos preparar o computador para funcionar como transmissor e receptor de Serial. Sendo assim utilizou-se o adaptador FTDI USB/UART para os testes.

9.1 Testes software TX e RX

Ao ligar o RX no TX do adaptador deveremos obter o mesmo dado enviado também recebido:

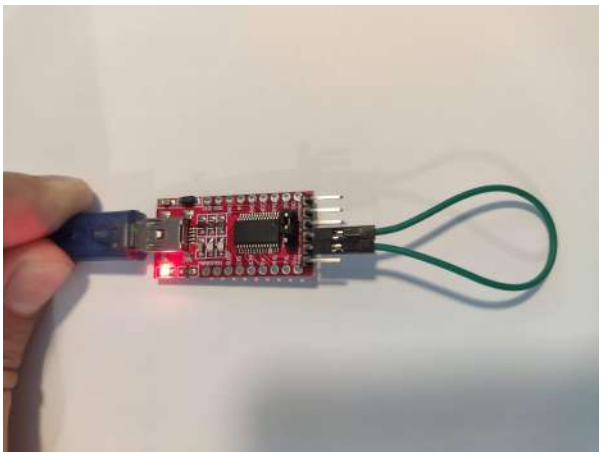

Figura 21: Testes com adaptador USB

Para preparar as permissões e obter a porta usou-se os seguintes comandos no terminal Linux.

```
sudo sysctl kernel.dmesg_restrict=0
//sudo sysctl kernel.dmesg_restrict=0
sudo sysctl kernel.dmesg_restrict=0
//[    0.103370] printk: console [tty0] enabled
//[    4.854103] usb 1-2: FTDI USB Serial Device converter now attached to
    ttyUSB0
sudo usermod -a -G tty mmc
sudo usermod -a -G dialout mmc
```

```
groups
//mmc adm tty dialout cdrom sudo dip plugdev lpadmin lxd sambashare
```

A transmissão UART será feita em dois softwares diferentes, um para envio e outro para recebimento.

A configuração da porta e velocidade é feita no arquivo **RS232Comm.h**:

```
#define BIT_RATE B9600

const char PORT1[] = "/dev/ttyUSB0";
const char PORT2[] = "/dev/ttyUSB0";

// Function prototypes
int openPort(const char* port);
void configPort(int fd);
```

Assim ao rodar o software de receptor e rodar o transmissor o terminal de recebimento receberá a mesma mensagem pois os pinos TX e RX do adaptador estão interligados. Sendo assim obteve-se sucesso até esta etapa.

Os outros arquivos podem ser encontrados no Github do projeto.

9.2 Inclusão do ESP32 como TX

Para testes iniciais ligou-se o pino TX0 do ESP32 na porta RX do adaptador.

O teste consiste em programar o ESP32 para envio de uma string "TESTE"seguida de um contador para visualizar no terminal.

O seguinte software foi escrito e compilado na Arduino IDE. Escolheu-se a placa e ao carregar no ESP32 teremos um transmissor via UART para os testes de software.

```
//TESTES COM TX ESP32 E RX ADAPTADOR
int i = 0;

void setup() {
  Serial.begin(9600); //CONFIGURA BAUD RATE DA UART
}

void loop() {
  Serial.print("teste"); //ESCREVE A MENSAGEM
  i++;
  Serial.println(i);   //IMPRIME O VALOR DO CONTADOR
  if(i > 9) i=0;
  delay(500);
}
```

A partir disto, ligou-se o adaptador ao ESP32 com jumpers e compilou-se o seguinte código que fará leitura continuamente da porta RX:

```
int main() {
```

```
  int fd2;                  // file descriptors
  int readin, readinTot = 0;         // Bytes read from Rx port
  char buffer[140];            // read buffer
  char* bufptr;               // buffer ponter

  while(1){     //DEIXA O SOFTWARE PRESO PARA EFETUAR LEITURA CONTINUAMENTE
  // Open and configure ports
  fd2 = openPort(PORT2);           // Port to read from (Rx)
  configPort(fd2);             // Apply settings

  // Read data from Rx port
  bufptr = buffer;
  // Read characters into our string buffer until get a carriage return \r or
     new line \n
  while((readin = read(fd2, bufptr, buffer + sizeof(buffer) - bufptr - 1)) >
     0) {
    bufptr += readin;
    readinTot += readin;
    if (bufptr[-1] == '\n' || bufptr[-1] == '\r') break;
  }
  *bufptr = '\0';  // Null terminate the string so we can printf() it

  printf("\nTotal bytes read: %d\n", readinTot);
  printf("\nReceived message: %s\n", buffer);
  }
  return(0);
}
```

O receptor receberá as strings enviadas pelo ESP32 e se houver sucesso, no terminal veremos a sapida enviada.

Algumas saídas do terminal Linux:

```
Total bytes read: 132
Received message: teste6
Total bytes read: 140
Received message: teste7
```

9.3 Implementação da fila ao software receptor:

A principio somente uma classe simples foi usada para implementar o acesso a UART, os métodos são para ativar, desativar e ler a UART.

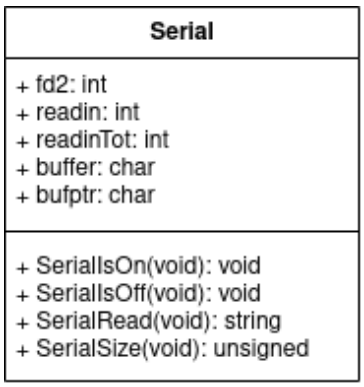

Figura 22: Classe Serial

No ESP32 gravou-se o seguinte código de teste:

```
//TESTES COM TX ESP32 E RX ADAPTADOR

int i = 0;

void setup() {
  Serial.begin(9600); //CONFIGURA BAUD RATE DA UART
}

void loop() {
    Serial.print("TESTE"); //ESCREVE A EMNSAGEM
    Serial.println(i);   //IMPRIME O VALOR DO CONTADOR
     i++;

     if(i==10)i=0;


  delay(500);
}
```

Primeiramente criou-se a classe para ativação, leitura e desativação do monitor Serial:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>  // Unix standard functions
#include <fcntl.h>  // File control funcitons
#include <errno.h>  // Error number definitions
#include <termios.h> // POSIX terminal control definitions
#include "RS232Comm.h"
#include <string>
using namespace std;

class Serial{
```

```
private:
  int fd2;                  // file descriptors
  int readin, readinTot = 0;         // Bytes read from Rx port
  char buffer[140];            // read buffer
  char* bufptr;              // buffer ponter
public:
  void SerialIsOn();
  void SerialIsOff();
  string SerialRead();
  unsigned SerialSize();
};




void Serial::SerialIsOn(){
  // Open and configure ports
  fd2 = openPort(PORT2);           // Port to read from (Rx)
  configPort(fd2);             // Apply settings

  // Read data from Rx port
  bufptr = buffer;
  // Read characters into our string buffer until get a carriage return \r or
      new line \n
  while((readin = read(fd2, bufptr, buffer + sizeof(buffer) - bufptr - 1)) >
      0) {
    bufptr += readin;
    readinTot += readin;
    if (bufptr[-1] == '\n' || bufptr[-1] == '\r') break;
  }
  *bufptr = '\0';  // Null terminate the string so we can printf() it
  SerialIsOff();
}

unsigned Serial::SerialSize(){
  return readinTot;
}

void Serial::SerialIsOff(){
    // Close the port(s)
  close(fd2);
}

string Serial::SerialRead(){
  return buffer;
}
```

E o seguinte código de teste para armazenar os dados enviados na UART pelo ESP32:

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include "Serial.h"
#include <list>

using namespace std;

unsigned i = 0;

int main() {
  Serial Serial1;

    list <string> test;
    int tam;
    tam=10;

    for(int i=0;i<tam;i++){
      Serial1.SerialIsOn();
      test.push_front(Serial1.SerialRead());
      cout << Serial1.SerialRead() << endl;
    }

    cout << "Tamanho da lista: " << test.size() << endl;

    tam = test.size();

    for(int i=0;i<tam;i++){
        cout << test.front() << endl;
        test.pop_front();

    }

    cout << "fim" << endl;

    return 0;
}
```

Que fornece a seguinte saída:

```
TESTE1
TESTE2
TESTE3
TESTE4
TESTE5
TESTE6
```

```
TESTE7
TESTE8
TESTE9

Tamanho da lista: 10
TESTE9
TESTE8
TESTE7
TESTE6
TESTE5
TESTE4
TESTE3
TESTE2
TESTE1

fim
```

10 Software Smartphone

A versão 2 inicia o desenvolvimento do software de comunicação entre ESP32 e Smartphone via Bluetooth

10.1 Bluetooth LE

Como já citado o sistema utilizará Buetooth LE q é uma especificação adotada pelo Bluetooth SIG em abril de 2009 que permite que periféricos de baixa potência com bateria de meses a anos se comuniquem com Bluetooth em aparelhos ou outros dispositivos. O Bluetooth LE abre uma nova gama de dispositivos e aplicativos, como sensores médicos no corpo e equipamentos esportivos e de condicionamento físico. O LE surgiu de uma tecnologia chamada Wibree (de propriedade da Nokia) e anteriormente era chamada de ultra-low power (ULP) Bluetooth.

Será utilizada a biblioteca fornecida pela Adafruit para bluetooth no ESP32 e portanto não haverá protocolos físicos entre dispositivos, apenas a comunicação RF entre smartphone e ESP32.

10.2 Testes via aplicativo terceiro

O aplicativo disponível na Play Store: Serial Bluetooth terminal permite a comunicação via Bluetooth utiliando diretamente acesso ao terminal do ESP32. Assim, fazendo o upload do código disponível na biblioteca e salvo no github podemos realizar o teste da funcionalidade do bluetooth:

```
//This example code is in the Public Domain (or CC0 licensed, at your option
    .)
```

```
//By Evandro Copercini - 2018
//
//This example creates a bridge between Serial and Classical Bluetooth (SPP)
//and also demonstrate that SerialBT have the same functionalities of a
    normal Serial

#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable
    it
#endif

#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Bluetooth not available or not enabled. It is only available
    for the ESP32 chip.
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

Assim comprova-se o desempenho da comunicação e a facilidad de programação utilizandos as bibliotecas fornecidas.

11 Machine Learning

Aqui serão desenvolvidas a parte de treinamento do modelo para previsao. iniciu-se a pesquisa e estou travado em um bug no google colab.

Para a implementação do modelo que fará a previsão de chuva ou não serão usados os sensores de temperatura, umidade e luminosidade conforme já citados.

Agora é necessário obter um banco de dados ou obter um com os próprios sensores

que serão usados.

Será necessário as seguintes etapas para utilização do modelo no ESP32:

Figura 23: Diagrama de processos Tensorflow Lite

O modelo será treinado no Google CoLab e salvo em Hexadecimal para ser movido para o Arduino. Os dados foram divididos em 90% para treinamento e 10% para teste.

11.1 Banco de dados obtido

Os dados foram amostrados a cada minuto, computados e carregados suavizados com médias de 15 minutos. Os dados são informações de tempo em UTC.

O banco de dados foi obtido do site Machine Learning Repository e conta com os seguintes dados:

1. Data: em UTC.
2. Hora: em UTC.
3. Temperatura interior (sala de jantar), em ºC.
4. Temperatura interior (ambiente), em ºC.
5. Temperatura de previsão do tempo, em ºC.
6. Dióxido de carbono em ppm (sala de jantar).
7. Dióxido de carbono em ppm (sala).
8. Umidade relativa (sala de jantar), em %.
9. Umidade relativa (ambiente), em %.
10. Iluminação (sala de jantar), em Lux.
11. Iluminação (sala), em Lux.
12. Chuva, a proporção dos últimos 15 minutos em que foi detectada chuva ([0,1]).
13. Pôr do sol.
14. Vento, em m/s.
15. Luz solar na direção oeste, no Lux.
16. Luz solar na direção leste, no Lux.
17. Luz solar na direção sul, no Lux.
18. Irradiância solar, em W/m2.
19. Motor entálpico 1, 0 ou 1 (liga-desliga).
20. Motor entálpico 2, 0 ou 1 (liga-desliga).
21. Motor turbo entálpico, 0 ou 1 (liga-desliga).
22. Temperatura exterior, em ºC.
23. Humidade relativa exterior, em %.
24. Dia da semana (calculado a partir da data), 1=Seg, 7=Dom.

Tabela 2: Características do banco de dados obtido - Original

Porém como já apresentado, os sensores usados serão de temperatura, luminosidade e umidade. Então o modelo considerará somente os sensores embarcados. Sendo assim os dados utilizados serão:

1. Data
2. Hora
3. Temperatura interior (ambiente/sala), em ºC.
4. Umidade relativa (ambiente/sala), em %.
5. Iluminação (ambiente/sala), em Lux.
6. Luz solar na direção oeste, no Lux.
7. Luz solar na direção leste, no Lux.
8. Luz solar na direção sul, no Lux.
9. Temperatura exterior, em ºC.
10. Umidade relativa exterior, em %.
11. Chuva, a proporção dos últimos 15 minutos em que foi detectada chuva ([0,1]).

Tabela 3: Características do banco de dados - Adaptado

Os dados podem ser acessados em adaptado.csv. Os dados foram adaptados e contam com valores booleanos para chuva, valores dos sensores, sem data nem hora.

12 Referencias

1. Notas de aula;
2. https://www.cmrr.umn.edu/ strupp/serial.html
3. http://www.tldp.org/HOWTO/Serial-Programming-HOWTO/index.html
4. http://man7.org/linux/man-pages/man3/termios.3.html
5. https://man7.org/linux/manpages/man2/fcntl.2.html