UNIVERSIDADE FEDERAL DE SANTA
CATARINA CAMPUS TRINDADE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Felipe Costa Juliano

Sistema embarcado estação meteorológica

Florianópolis
2022

Felipe Costa Juliano

Sistema embarcado estação meteorológica

Relatório submetido à disciplina Programação de Sistemas Embarcados da Universidade Federal de Santa Catarina como requisito parcial à obtenção de grau referente ao semestre 2022.2.
Orientador: Prof. Eduardo Augusto Bezerra, Dr.

Florianópolis
2022

RESUMO

Desenvolver um software na linguagem de programação c++ para o funcionamento de uma estação meteorológica, tal sistema necessita funcionar em um dispositivo mobile e em um computador.

Palavras-chave: c++, stm32f4, linux, estação meteorológica.

SUMÁRIO

1 INTRODUÇÃO

O presente relatório tem como objetivo documentar a implementação do código de um controlador de uma estação meteorológica, de acordo com as especificações solicitadas.

1.1 REQUISITOS DO PROGRAMA

A estação capta dados como temperatura, velocidade do vento e umidade, o *software* também precisa utilizar um modelo de aprendizado de máquina para que seja possível economizar bateria e tempo de processamento do dispositivo.

2 DESENVOLVIMENTO

Inicialmente, foram levantados os requisitos de funcionamento para o projeto. Definiu-se a necessidade de uma máquina de estados para o início do projeto.

Figura 1 – Máquina de Estados Finita.

Comandos de entrada e ciclos							
Estado atual	**Nada**	**Botão start**	**Nada**	**Nada**	**Nada**	**Reinicia o ciclos**	**Botão Stop**
Captando funcionamento dos sensores	Software Aguardando start	Iniciando ciclo	Finalizado	Finalizado	Finalizado	Iniciando ciclo	Desligado
Capitando dados	Software Aguardando start	Aguardando ciclo anterior	Iniciando ciclo	Finalizado	Finalizado	Aguardando ciclo anterior	Desligado
Analisando dados	Software Aguardando start	Aguardando ciclo anterior	Iniciando ciclo	Finalizado	Finalizado	Aguardando ciclo anterior	Desligado
Comparando dados	Software Aguardando start	Aguardando ciclo anterior	Aguardando ciclo anterior	Iniciando ciclo	Finalizado	Aguardando ciclo anterior	Desligado
Escrevendo log	Software Aguardando start	Aguardando ciclo anterior	Aguardando ciclo anterior	Iniciando ciclo	Finalizado	Aguardando ciclo anterior	Desligado
Realizando ou não o envio de dados	Software Aguardando start	Aguardando ciclo anterior	Aguardando ciclo anterior	Aguardando ciclo anterior	Iniciando ciclo	Aguardando ciclo anterior	Desligado

Fonte: Elaborado pelos autores (2022).

Todo o código será construído tomando-a como base, os únicos comandos permitidos são *start* para iniciar o ciclo e o *stop* que interrompe o ciclo do sistema.

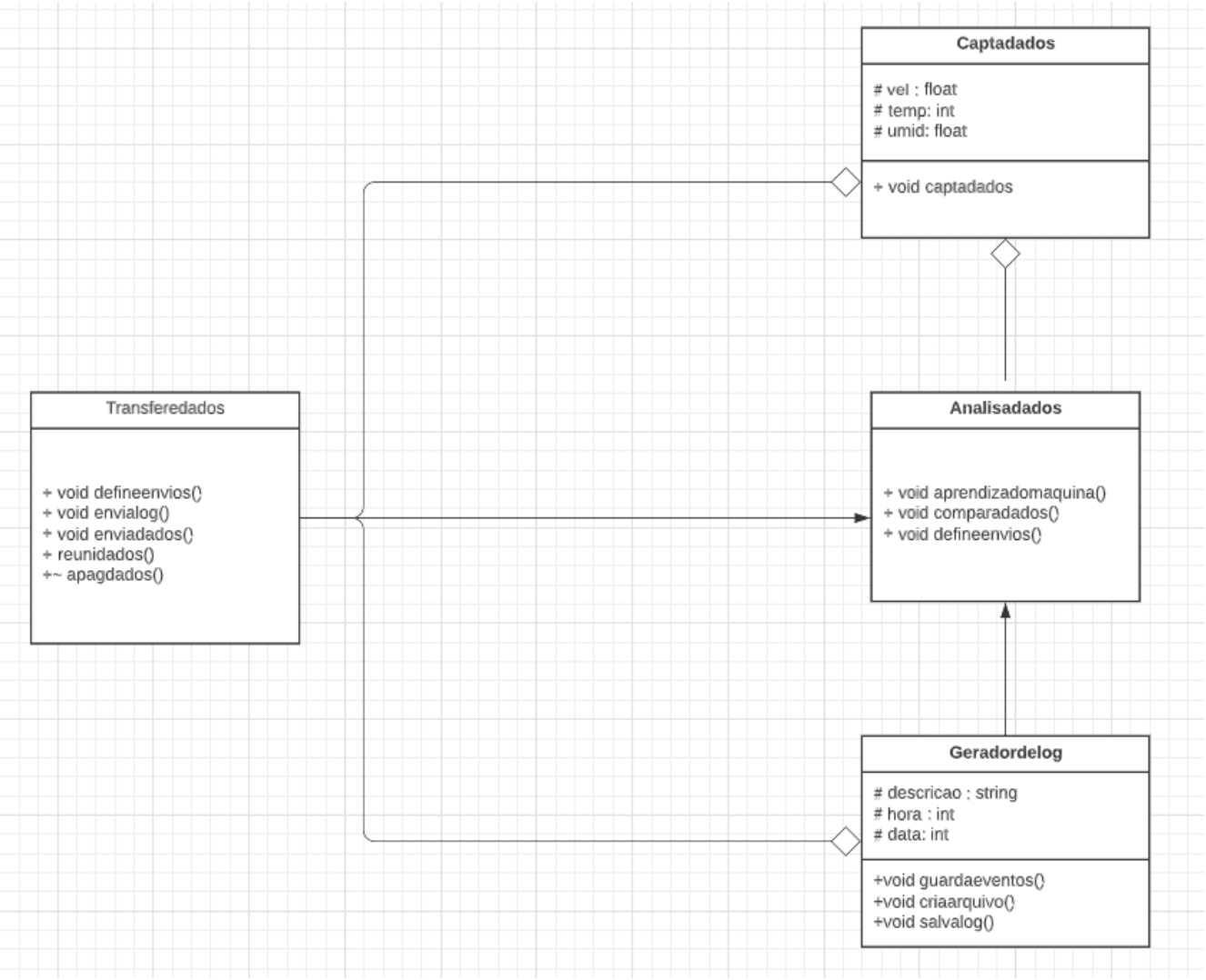
2.1 DIAGRAMA DE CLASSES

O digrama de classes, mostrado na Figura 2, foi criado visando o funcionamento do programa na plataforma STM32F4 visando os conceitos aprendidos em sala, as classes empregadas são: *transferedados; captadados; analisadados e geradordelog.*

2.1.1 Classe Captadados

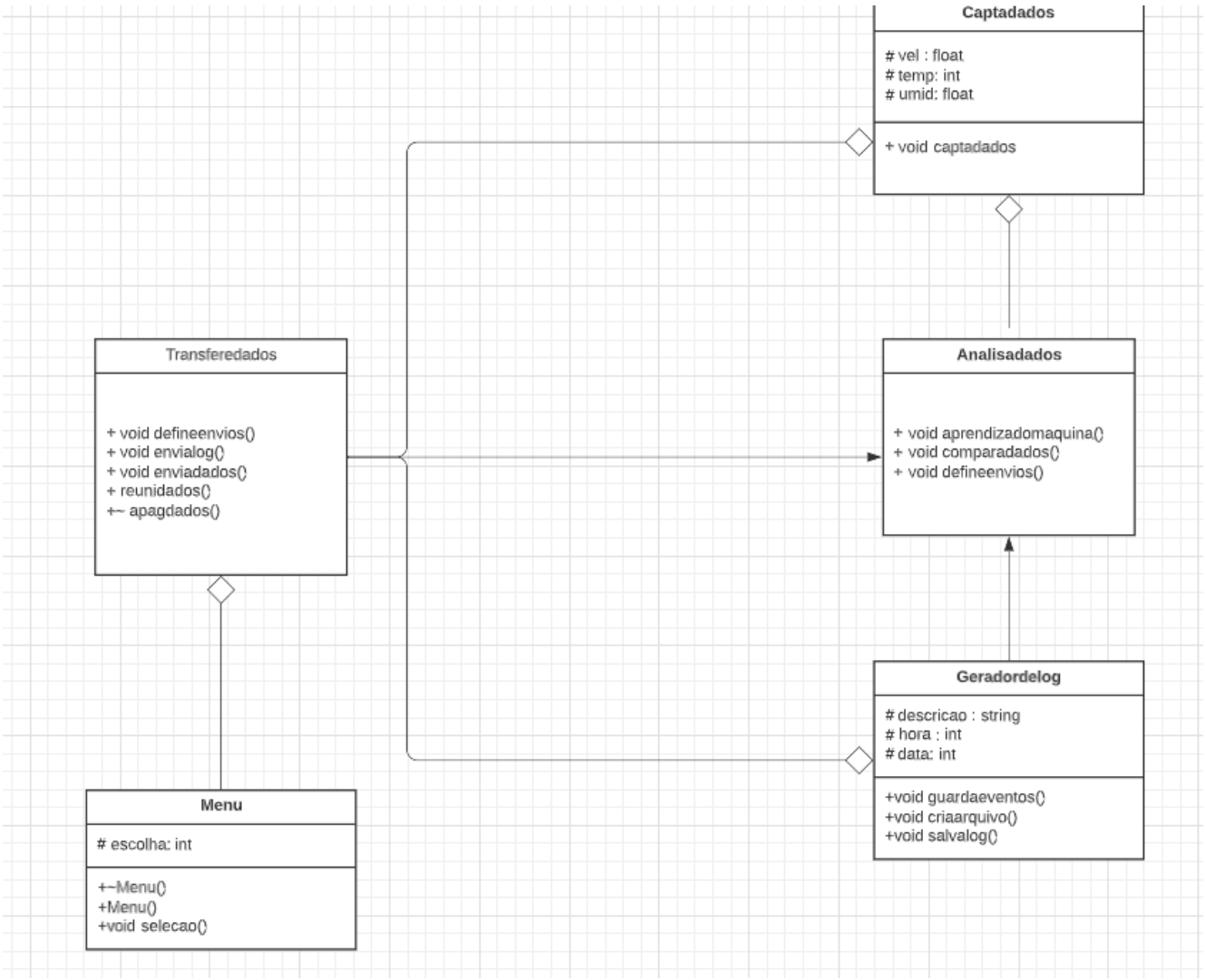
A classe *captadados* está direcionada apenas em captar os dados e armazenar em variáveis temporárias, utilizando apenas três variáveis, pois os dados captados são apenas em três grandezas.

Figura 2 – Diagrama de classes para o microcontrolador.



Fonte: Elaborado pelo autor (2022).

Figura 2 – Diagrama de classes do projeto para o software no computador.



Fonte: Elaborado pelo autor (2022).

2.1.2 Classe *Analisadados*

Na classe *analisadados* será implementado o funcionamento do aprendizado de máquina, aqui o sistema poderá prever alguns dados dos sensores e se a previsão for próxima ao dado real o método *defineeventos* não permitirá o envio de dados.

2.1.3 Classe *Geradordelog*

Estará responsável pelo armazenamento de *logs* do sistema, tal função só ocorre uma vez ao final de cada ciclo, como indicado na máquina de estados.

3 Classe *tranferedados*

Classe responsável pelo envio de dados, contém um construtor e um destrutor, chamados em ordem, *reunidados* e *apagadados*, é a principal classe do sistema, pois nela se reune dados das outras classes para realizar ou não o envio de dados.

4 Descrição de *hardware* e *software*

A placa utilizada será o *kit* de desenvolvimento da *STM* com o microcontrolador *STM32F4*, que dentre as opções disponíveis é um sistema de baixo consumo, atendendo assim um dos principais requisitos para o funcionamento do projeto. Os sensores que acompanharão o *kit* de desenvolvimento será um *DHT11*, que por sua vez é um sensor de baixo custo e de conhecimento bastante difundido, facilitando assim sua implementação, podemos utilizar um anemômetro simples para a medição de velocidade, apenas com esses dois sensores conseguimos implementar todas as medições propostas no sistema. No software para o computador foi implementado a biblioteca *CppLinuxSeria*, que auxilia na utilização da porta serial "https://github.com/gbmhunter/CppLinuxSerial".

No microcontrolador utilizaremos a *DHTLib* para o sensor de temperatura e umidade, para o anemômetro não é necessário utilizar bibliotecas, podemos fazer a implementação diretamente no código principal.

Apêndice A
Código Fonte

```
#include <iostream>
#include <CppLinuxSerial/SerialPort.hpp>

using namespace mn::CppLinuxSerial;

class Menu
{
   int escolha;
public:
   Menu();
   ~Menu();
   void selecao();


};

Menu::Menu()
{
   std::cout <<"=========================================\n";
   std::cout <<"         Estacao Meteorologica\n";
   std::cout <<"=========================================\n";
  }

Menu::~Menu()
{
   std::cout << "\nEncerrando programa\n";
}
  void Menu::selecao()
{


  std::cout <<" Selecione 1 para captar dados\n 0 para sair";
  std::cin >> escolha;

       if ((escolha = 1))
       {

          std::cout << "Captando dados";
                              SerialPort    serialPort("/dev/ttyUSB0",    BaudRate::B_9600,
```

```
NumDataBits::EIGHT, Parity::NONE, NumStopBits::ONE);
        // Use SerialPort serialPort("/dev/ttyACM0", 13000); instead if you want to provide a custom baud rate
        serialPort.SetTimeout(-1); // Block when reading until any data is received
        serialPort.Open();

        // Read some data back (will block until at least 1 byte is received due to the SetTimeout(-1) call above)
        std::string readvel;
        serialPort.Read(readvel);

        std::string readtemp;
        serialPort.Read(readtemp);

        std::string readData;
        serialPort.Read(readData);


       std::cout << readvel;
       std::cout << readtemp;
       std::cout << readData;

        // Close the serial port
        serialPort.Close();

      }
        if((escolha = 0)){
          std::cout << "\nReiniciando sistema\n";
          main();


        }
        else ( escolha != 1 & escolha != 0){

          std::cout << "\nDigite uma entrada válida\n";
          main();


       }
       }

      int main() {
```

```
#if defined __linux__
    cout << "Ola, Linux!" << '\n';

   Menu *ponteiro;

ponteiro = new Menu();

ponteiro -> selecao();

#elif _WIN64
    cout << "Troque o sistema operacional" << '\n';

#elif _WIN32
    cout << "Troque o sistema operacional" << '\n';

#else
    cout << "Troque o sistema operacional" << '\n';
#endif



}
```