

Bússola eletrônica de 2 eixos

Gustavo Batistell¹

¹UFSC - Universidade Federal de Santa Catarina ,
Curso de Graduação em Engenharia Eletrônica

6 de dezembro de 2023

1 Introdução

O objetivo deste projeto é desenvolver um magnetômetro eletrônico capaz de apresentar, entregar e salvar os dados de *heading*, ou direção em relação ao campo magnético terrestre, ou seja, a direção apontada pelo equipamento. Os magnetômetros eletrônicos são parte importante na instrumentação de embarcações, aeronaves, submarinos, satélites, etc e durante a realização do estágio no Laboratório do Grupo Drakkar Atlantec em Florianópolis, foi constatada a demanda para dominar tanto o ajuste quanto a calibração deste tipo de instrumento. Apesar da empresa contar com as boninas de Helmholtz com controlador e um magnetômetro de referência, tanto a parte eletrônica quanto os softwares embarcados carecem de atualização, melhoria e nova implementação para completo domínio do funcionamento. Essa necessidade de dominar (novamente) por completo o sistema surgiu devido perda de documentação e dos códigos originais com o passar dos anos.

Para isso, a tarefa do desenvolvimento inicial foi dividida em duas partes: uma parte para acionamento (driver) e outra parte para um magnetômetro, este último no qual fiquei incumbido.

Assim, utilizado um módulo micro-controlado ARM RP2040 Raspberry Pi Pico [1], um módulo magnetômetro para medir o campo magnético em pelo menos 2 eixos (seja o HMC5883 ou HoneyWell HMR3300 [2] disponíveis no laboratório), realizar o acionamento de leds para apresentar o *heading*, realizar log de dados e suportar acesso via UART/USB e Wireless. Ver diagrama da Figura 1.

2 Requisitos

Os requisitos de projeto foram combinados com a demanda do Laboratório do Grupo Drakkar Atlantec e as especificações solicitadas pelo professor, aproveitando-se da portabilidade e modularidade com a linguagem de programação C++ e os concei-

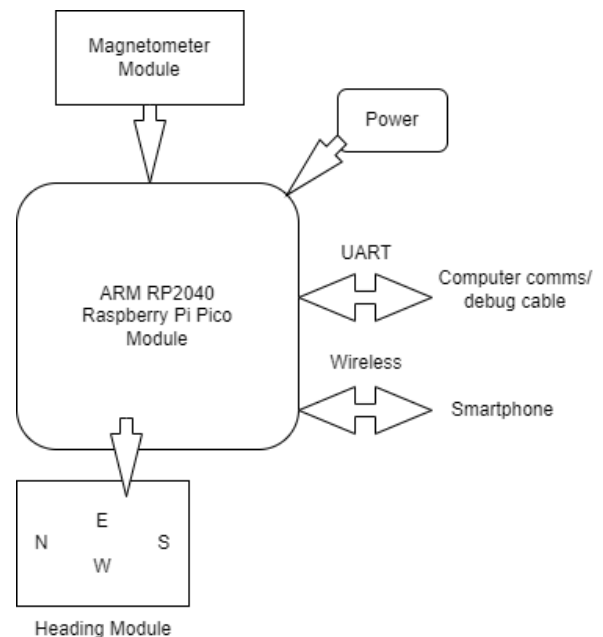


Figura 1: Diagrama de blocos revisado do projeto.

tos de orientação a objetos, polimorfismo, classes, etc. Assim, foram listados os requisitos de desenvolvimento combinados para uma Bússola Eletrônica conforme segue:

- Apresente em mostrador simples de LEDs o *heading* em 8 direções: N, NE, E, SE, S, SW, W e NW.
- Utilize um módulo de magnetômetro de pelo menos 2 eixos (seja analógico ou digital) para obtenção das componentes de campo magnético.
- Com as componentes X e Y de campo magnético calcular a direção (em graus de 0 a 360) que também pode ser utilizada para o *heading*.
- Realizar log da ID, timeStamp e dados.

- Aceitar a consulta e envio deste log para um *host*.
- Aceitar comunicação via UART/USB.
- Aceitar comunicação *wireless*.
- O *host* pode conferir a listagem do log.
- O *host* pode conferir o tempo em que a Bússola Eletrônica se manteve ativa

3 Design Proposto do sistema embarcado

Como o microcontrolador ARM RP2040 [1] é *dual-core* foi decidido implementar 2 máquinas de estado. Uma delas para a aplicação da Bússola Eletrônica propriamente dita, que ao ser iniciada segue diretamente para o estado de aquisição de dados brutos, depois para um estado que calcula a direção a partir das componentes X e Y dos dados brutos, outro estado que armazena o log e finalmente um estado que exibe o *heading*. Essa primeira máquina de estados na *Thread 1* é apresentada na Figura 2 e é importante destacar a importância dela na gravação/escrita em RAM. Já a segunda máquina de es-

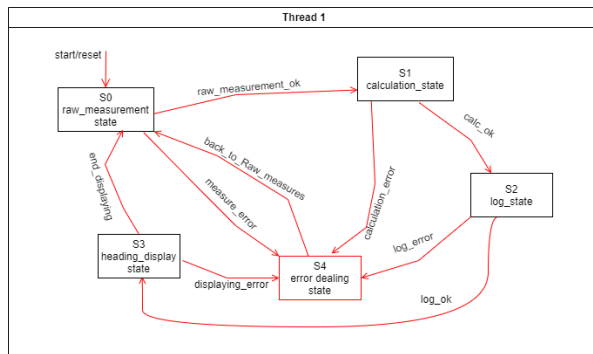


Figura 2: Máquina de estados principal (com gravação em memória).

tados na *Trhead 2* é incumbida de controlar a conectividade UART e *Wireless*, conforme mostrado na [Figura 3](#). Esta máquina de estados tem um estado de *idle* para quando não houver conexão, um estado para conexão *wireless* estabelecida, um estado para o `wirelessRequest`, um estado para quando a conexão UART é estabelecida, um estado para o `UARTrequest` e um estado para tratamento de erros. Esta máquina desta *Trhead 2* apenas acessa os dados. Essa distinção entre gravação e acesso deste projeto em específico é importante para evitar problemas de leitura e escrita usuais em programação concorrente quando mais de uma *thread* é possível.

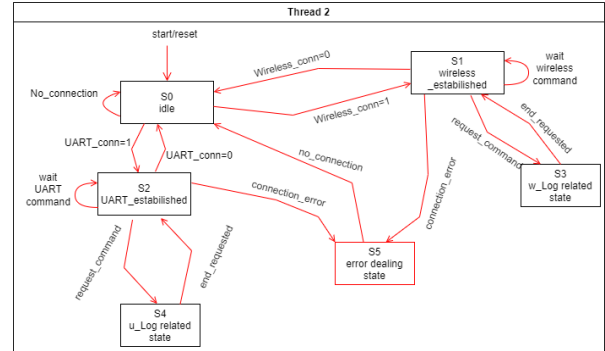


Figura 3: Máquina de estados secundária para conexões UART e Wireless (apenas acesso a memória).

Para garantir a modularidade do projeto, o diagrama de classes proposto utilizou de 5 níveis gerais (com uma subdivisão para a camada de drivers externos e *on-board* incluída em revisão) de abstração que são listados abaixo em uma abordagem "de dentro para fora":

- **Hardware Abstraction Layer (HAL):** Responsável por abstrair o hardware subjacente, incluindo os módulos do SDK da Raspberry Pi Pico.
- **Device Drivers Layer:** Essa camada mais geral foi subdividida após revisão, pois gerencia os drivers para GPIO, UART, comunicação sem fio (que são *on-board* da Raspberry Pi Pico) e o magnetômetro (que é um dispositivo externo à Raspberry Pi Pico, por isso locado na subcamada de drivers externos).
- **Processing Layer:** Lida com o processamento dos dados, incluindo a classe ‘DirectionCalculator’. Esta camada também pode ser usada para implementações futuras de técnicas de Digital Signal Processing como média móvel, digitalização, etc.
- **Control/Application Layer:** Responsável por controlar as partes funcionais do projeto. É o ponto central onde as operações funcionais são coordenadas, incluindo as classes ‘HeadingDisplay’, ‘Log’, ‘UART’ e ‘Wireless’.
- **User Interaction Layer:** É a camada mais externa que lida com a interação do usuário, onde está a classe “ElectronicCompass” propriamente dita.

Essa organização se apresentou clara e permite uma separação adequada de responsabilidades em cada camada de classes do sistema, o que é uma boa prática para melhor documentar, estruturar e facilitar a compreensão e manutenção do projeto. O diagrama de classes pode ser visto na [Figura 4](#) do no Apêndice A.

4 Design Proposto para o *Host*

A proposta para o *Host* é implementar um software que deve rodar no computador, na máquina virtual WSL Ubuntu, apresentar um menu na tela em que um usuário ADMIN possa: escolher algumas das opções apresentadas pressionando a tecla específica no teclado+<enter>; o software envia pela USB (porta COM) e recebe comandos e dados, respectivamente, do sistema embarcado externo a esse computador que usa a UART a 115200 kbps.

Quando o software é iniciado, o usuário ADMIN deve confirmar o início da conexão, e pode escolher da lista do menu:

- listar e pegar todos os eventos ocorridos em um determinado intervalo de datas;
- obter o tempo total em hh:mm que o controlador esteve ativo;
- listar e pegar os eventos instantâneos (*real time*);
- desativar conexão/sair.

Usando o mesmo conceito de camadas para o projeto do software do *Host*, o diagrama de classes é apresentado na [Figura 5](#) do Apêndice B. Dessa maneira, a modularidade do projeto no *Host* também é facilitada, por exemplo se houver interesse em implementar o sistema para outro módulo como STM32 ao invés da Raspberry Pi Pico. A descrição das camadas é parecida com o que já foi descrito para o projeto embarcado, de dentro para fora:

- Device Drivers Layer: Essa camada mais geral gerencia os drivers para comunicação com a Raspberry e o driver da serial port.
- Control/Application Layer: Responsável por controlar o equipamento e os dados/comandos de saída/entrada. É o ponto central onde as operações funcionais são coordenadas.
- User Interaction Layer: É a camada mais externa que lida com a interação do usuário, e sua interface com o software.

5 Resultados de implementação

A execução do projeto como um todo levou em consideração o uso das técnicas apresentadas em aula. Os exemplos apresentados a seguir constam em trechos dos códigos disponíveis no repositório do GitHub[3].

5.1 Uso de Herança e Herança Múltipla

No decorrer do projeto o uso de Herança foi essencial, por exemplo nas classes Connection e Pico-Connection do projeto do *Host*, mas certamente o exemplo mais notável é de Herança Múltipla nas classes base Clock e Calendar e na classe derivada ClockCalendar. A classe ClockCalendar herda publicamente tanto de Clock quanto de Calendar, indicando a Herança Múltipla, como pode ser visto no trecho de código abaixo:

```
#include "ClockCalendar.hpp"

ClockCalendar::ClockCalendar(int t_day, int t_month,
                             int t_year, int t_hr, int t_min, int t_sec,
                             int t_isPm)
    : Clock(t_hr, t_min, t_sec, t_isPm), Calendar(t_day,
                                                  t_month, t_year) {
    // Inicializa classe base Clock c.parametros de tempo
    // Inicializa classe base Calendar c.parametros de data
}

void ClockCalendar::advance() {
    // Implementacao de advance() para classe ClockCalendar
    // chamar advance() das classes base Clock e Calendar
}
```

5.2 Uso de Friends

Um exemplo de uso de Friends, que é uma forma de permitir que uma classe ou uma função acesse os membros privados ou protegidos de outra classe, sem violar o princípio do encapsulamento, foi utilizado nas classes Connection e Data do projeto do *Host*. Essa implementação pode ser vista nos códigos fonte dos arquivos Connection.h, Data.h e Data.cpp. Abaixo segue recorte do arquivo Connection.h mostrando o uso de Friends (notar que trechos do código considerados irrelevantes para o exemplo foram omitido com "..."):

```
/**
 * *****
 * File:      Connection.h
 * ...
 * public:
 * ...
 * friend class Data;
 * //Declara classe Data como amiga para
 * //acessar membros privados de Connection
 *
 * queue<string> dataQueue;
 * // Fila para armazenar dados recebidos pela serial
 * // declarada como pública para classe Data acessar
 * ...
 */
```

5.3 Uso de Template

Um template pode ser útil para evitar a repetição de código ou para implementar algoritmos independentes de tipos. Um exemplo de uso de template pode ser visto no projeto do *Host*, especificamente na classe Data, que usa um template para definir o método chamado saveData, que salva os dados da fila dataQueue em um arquivo de texto, com qualquer tipo de dado que possa ser convertido em string. Segue trecho do código citado:

```
...
template <typename T>
void Data::saveData(queue<T> dataQueue) {
    ofstream file;
    // Cria objeto classe ofstream para saida
    file.open("data.txt");
}
```

```

// Abre o arquivo data.txt

if (file.is_open()) {
    // Se aberto com sucesso

    while (!dataQueue.empty()) {
        // Enquanto fila dataQueue não vazia

        file << dataQueue.front() << endl;
        // Escreve elemento da frente da
        //fila no arquivo com quebra de linha

        dataQueue.pop();
        // Remove o elemento da frente da fila
    }
    file.close();
    // Fecha o arquivo

    cout << "Dados salvos com sucesso." << endl;
}
else {
    // Se arquivo não aberto
    cout << "Erro ao abrir o arquivo." << endl;
}
...
}

```

5.4 Uso de Funções Virtuais

As funções virtuais são funções ou métodos cujo comportamento pode ser sobrescrito em uma classe herdeira por uma função com a mesma assinatura e são parte fundamental para funcionamento do conceito de Polimorfismo Abstrato. Um exemplo de uso de Funções virtuais no projeto do *Host* pode ser visto nos códigos dos arquivos *Connection.h* e *PicoConnection.h* abaixo, respectivamente. Abaixo seguem recortes dos arquivos mostrando o uso (notar que trechos do código considerados irrelevantes para o exemplo foram omitido com "..."):

```

...
#include <string>
#include "PicoConnection.h"
// Arquivo cabeçalho da classe PicoConnection

#include "SerialPort.h"
// Arquivo de cabeçalho da classe SerialPort
#include...

...
class Connection {
private:
...
public:
...
    virtual void send(string message);
    // para enviar mensagem pela serial

    virtual string receive();
    // para receber mensagem pela porta serial
}

...

#include <string>
#include "Connection.h"
// arquivo de cabeçalho da classe Connection
...

class PicoConnection
...
private:
...
public:
...
    void send(string message) override;
    // para enviar comando pela serial

    string receive() override;
    // para receber resposta pela serial
...
...

```

5.5 Uso do Tratamento de Exceções

Tratamento de exceções é um recurso que permite lidar com situações de erro ou anormais e sua aplicação consiste no uso de três palavras-chave: *try*, *throw* e *catch*. No trecho de código abaixo da classe *DirectionCalculator* do projeto embarcado é apresentada uma solução pra tratamento da exceção de divisão por zero:

```

// Metodo para calcular direcao com dados brutos
void DirectionCalc::calculateDirection() {

    // Simulacao de calculos
    rawX = 1.0;
    rawY = 2.0;

    // bloco try-catch para tratar a execucao
    try {

        // Verifica se os dados brutos sao validos
        if (rawX <= 0 || rawY <= 0) {

            // Lanca uma execucao com a mensagem
            throw "Dados brutos inválidos";
        }
        else {

            // Realiza calculos para obter a direcao
            // (trecho em implementação)
            double calculatedDirection = rawX + rawY;

            // Atualiza o log com dados
            updateLog("Direction calculated: "
                + std::to_string(calculatedDirection));
        }
    }
    catch (const char* msg) {

        // Captura a execucao se for uma string
        // Mostra a mensagem de erro
        cerr << "Erro: " << msg << endl;
    }
}

int main() {

    // Cria objeto da classe Log
    Log log;

    // Cria objeto DirectionCalc e passa log como parametro
    DirectionCalc dc(log);

    // bloco try-catch para tratar a execucao
    try {

        // Chama o metodo calculateDirection do objeto dc
        dc.calculateDirection();
    }
    catch (const char* msg) {

        // Captura a execucao se for uma string
        // Mostra a mensagem de erro
        cerr << "Erro: " << msg << endl;
    }
    catch (...) {

        // Captura qualquer outra execucao
        // Mostra mensagem erro
        cerr << "Erro desconhecido" << endl;
    }
    return 0;
}
...

```

5.6 Uso de sobrecarga de operadores

Um exemplo de sobrecarga do operador de inserção «» (que pode ajudar a organizar a estrutura e comprimento do código) foi utilizado para a inserção de registro à fila da classe *Log* do projeto embarcado:

```

#ifdef LOG_H

```

```

#define LOG_H

#include <queue>
#include <string>
#include "Compass.h" // Inclui classe Compass
#include "DirectionCalc.h" // Inclui classe DirectionCalc
#include "UART.h" // Inclui classe UART/USB
#include "Wireless.h" // Inclui classe Wireless

class Log {
public:
    // Construtor aceita objetos relacionados
    // como parametros
    Log(Compass& compass, DirectionCalc& directionCalc,
        UART& uart, Wireless& wireless);

    // Sobrecarga do operador << para adicionar log fila
    Log& operator<<(const std::string& logData);
}

```

5.7 Uso de Polimorfismo com Classes Abstratas

O Polimorfismo é um princípio fundamental na Programação Orientada a Objetos e envolve o uso conjunto de Herança e Funções Virtuais, permitindo que classes derivadas de uma única classe base invoquem métodos com a mesma “assinatura”, mas que se comportam de maneira diferente para cada uma das classes derivadas. De fato, neste projeto o uso de funções virtuais e herança, caracterizam o emprego de polimorfismo com classes abstratas. Por exemplo na classe base *Connection* e classe derivada *PicoConnection* do projeto do *Host* já exemplificadas acima na subseção “Uso de Funções Virtuais”.

5.8 Implementação de lista e fila (*Queue*)

Os eventos da fila com ID do sistema embarcado, data/hora do evento usando o *ClockCalendar* foram implementados na classe *Log* do projeto embarcado e a lista do projeto do *Host* já teve implementação apresentada acima no subitem Uso de Template, onde também foi implementada a limpeza da fila quando log é transferido para o *host*.

5.9 Implementação da Wireless

Apesar de dessa parte do projeto não ser mais requisito, ela tem menção em vários trechos do projeto e facilmente pode ser retomada utilizando o próprio *WirelessDriver* disponível em algumas versões da Raspberry Pi Pico, entrando na parte do projeto embarcado na camada Onboard Drivers Layer. A outra “ponta” da comunicação wireless pode ser um smartphone Android com software projetado nos mesmos moldes do projeto de software do *Host*.

5.10 Montagem dos módulos/circuitos

A montagem dos módulos foi realizada em protoboard conforme a [Figura 6](#) do Apêndice C. Mais

a esquerda na imagem encontra-se o módulo magnetômetro HoneyWell [2]. Foi implementado circuito com transistores para ajustar as tensões de 3.3V e 5V entre as conexões SPI do módulo magnetômetro [2] e a Raspberry Pi Pico [1]. Mais ao centro da imagem está o módulo Raspberry Pi Pico [1] rodeado pelos 8 LEDs previstos para indicação da direção (*heading*). Mais a direita da imagem o módulo em vermelho está um conversor UART-USB da FTDI.

6 Conclusões

A organização do projeto de sistemas embarcados por blocos funcionais e, principalmente, por diagrama de classes disposto em camadas, é certamente uma das principais vantagens constatadas no decorrer deste trabalho. A abstração necessária para isso pode tomar certo tempo para ser desenvolvida, mas garante vislumbrar uma estrutura de organização modular para um trabalho colaborativo, fracionado/compartimentado, restringindo problemas a abrangência de problemas ou soluções de implementação a blocos de fácil identificação. Apesar da última versão do código do projeto não estar funciona no momento, apresenta organização, estrutura e andamento considerável para que em poucos dias de estágio já haja uma versão funcional dessa bússola eletrônica. A documentação também ficou bem didática e completa para que qualquer membro da equipe possa replicar resultados, desenvolver algum bloco funcional e colaborar de maneira online na ferramenta de controle de versão.

Cabe ressaltar que, em verdade a, a maior dificuldade de entendimento foi o uso desta ferramenta de controle de versão Git. A navegação para trabalhar em cada *branch* e as “imagens no tempo” para os *commits* e *merges* do repositório no GitHub eram contra intuitivos para quem estava habituado a organizar as idéias em pastas. Algumas dificuldades surgiram por exemplo com o resultado do descuido ao mover/copiar arquivos do Windows à máquina virtual WSL, onde surgiam arquivos incompativeis com os compiladores provocando erros diversos, especialmente nos *commits*. Por outras vezes o problema era como listar cuidadosamente os arquivos a incluir no *.gitignore*, etc.

Por fim o saldo é muito positivo: se sentir imerso num ambiente de trabalho voltado para projetos de sistemas embarcados parecia muito distante da realidade de quem sempre priorizou trabalhar e estudar a eletrônica analógica.

Referências

- [1] R. Pi. Rp2040 datasheet a microcontroller by raspberry pi. [Online]. Avail-

lable: <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf>

- [2] Honeywell. Digital compass solutions. [Online]. Available: https://aerospace.honeywell.com/content/dam/aerobt/en/documents/learn/products/sensors/datasheet/N61-2047-000-000_Digital_Compass_Solutions_HMR3300-ds.pdf
- [3] G. Batistell. Electroniccompass. [Online]. Available: <https://github.com/gustavobtt/ElectronicCompass.git>

Apêndice A

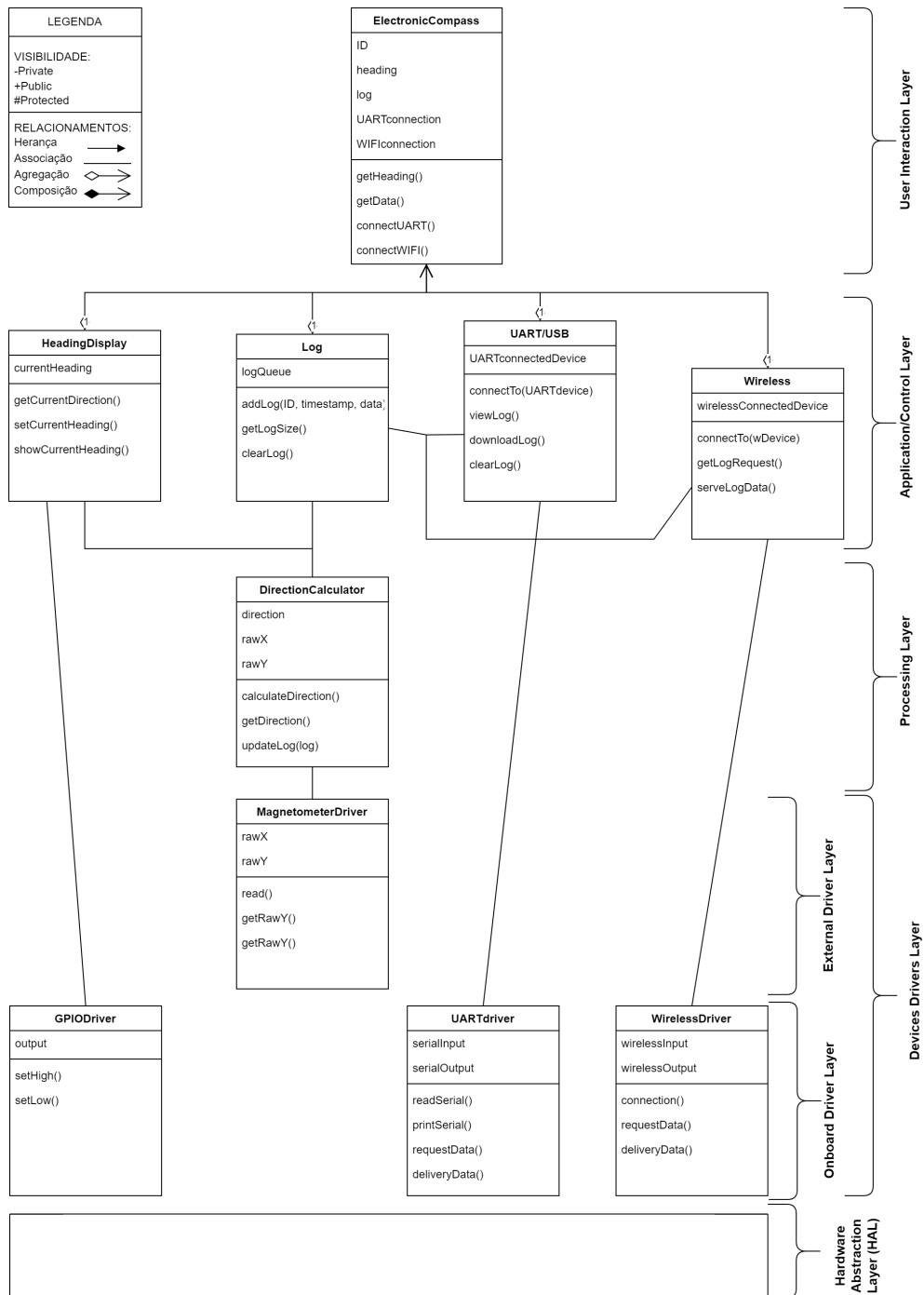


Figura 4: Diagrama de classes revisado do projeto organizado em camadas.

Apêndice B

Diagrama de classes do projeto do *Host*

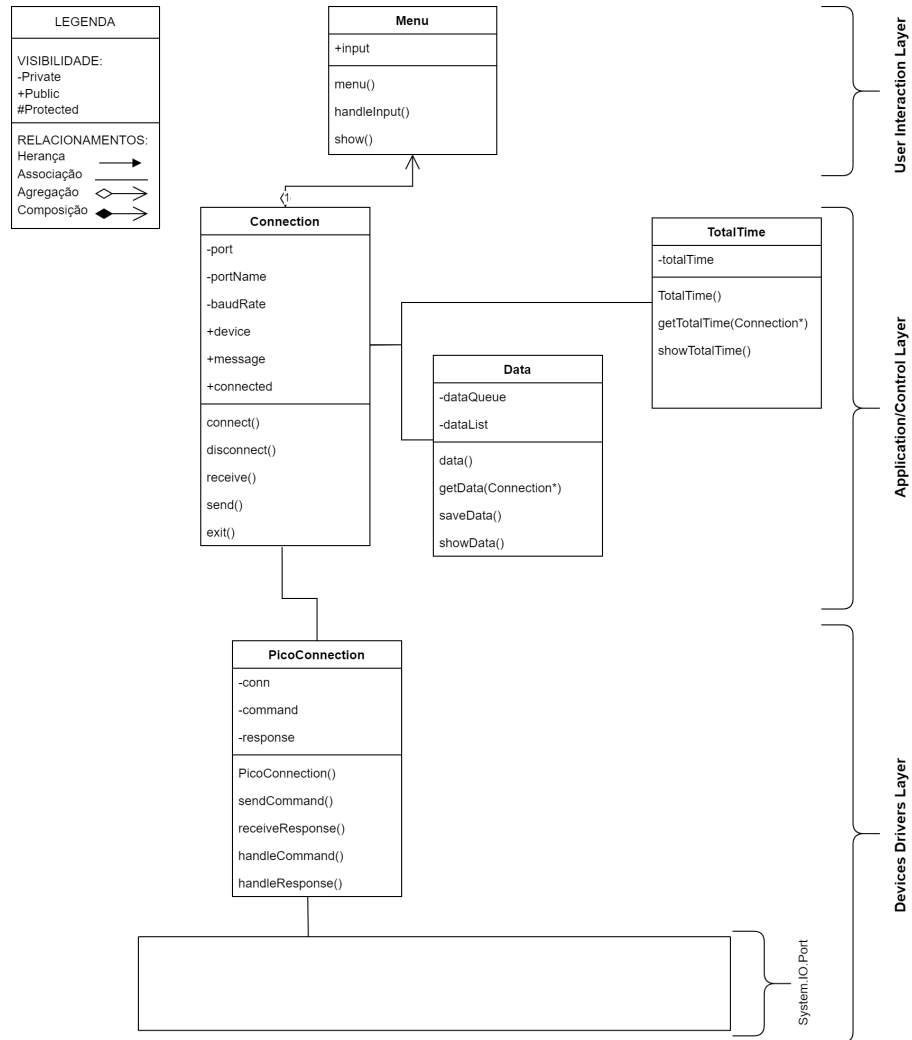


Figura 5: Diagrama de classes do projeto do *Host*.

Apêndice C

Montagem dos módulos do Projeto

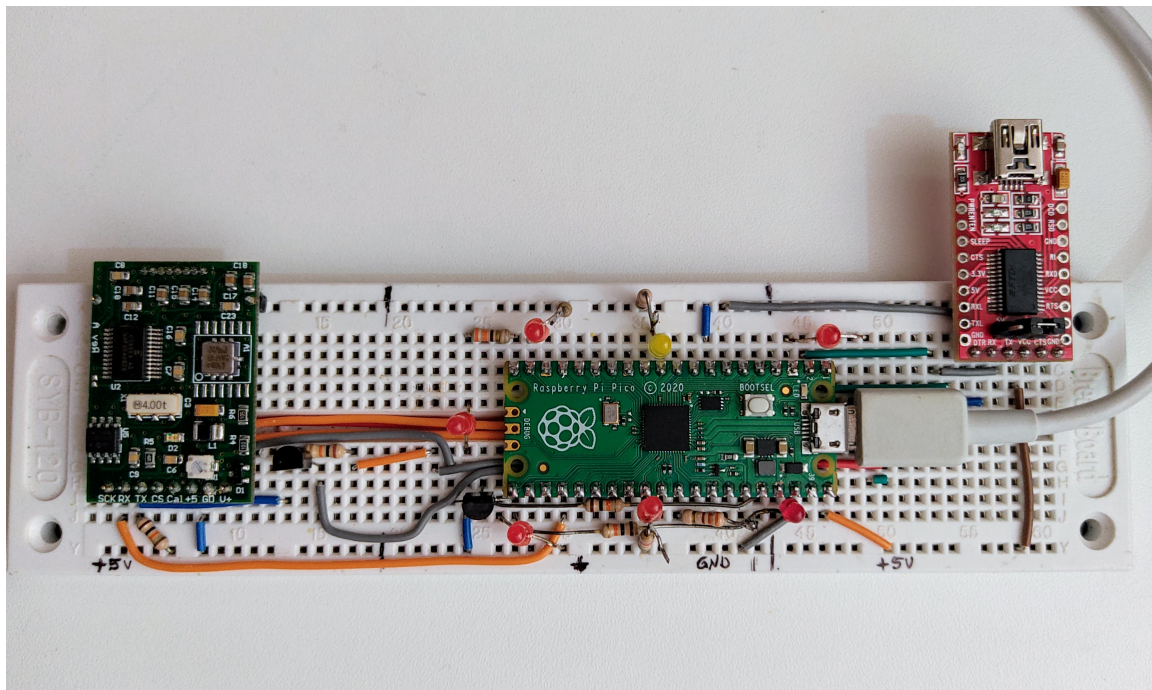


Figura 6: Montagem dos módulos do projeto.

Apêndice D

Plano de testes da Bússola eletrônica de 2 Eixos

Filename	<u>Electronic Compass</u>
Author	<u>Gustavo Batistell</u>
Application	<u>Bússola eletrônica de 2 Eixos</u>

Revision Log:

Date	Author	Version	Comments
29-Nov-2023	Gustavo <u>Batistell</u>	1	Versão inicial

Introdução

O objetivo desse plano de teste é orientar o teste de funcionamento do sistema Bússola Eletrônica de 2 Eixos e suas partes.

Ambiente

Uso de computador tipo PC rodando máquina virtual WSL Ubuntu no sistema operacional Windows 10 ou 11.

Processo

- Ligar o computador
- Rodar o WSL
- Clonar o repositório [3] para a pasta local HOME.
- Entrar na pasta /ElectronicCompass\$
- Rodar a aplicação com o comando pcHost

Testes 1

Resultados esperados:

- Obter sucesso na conexão da comunicação
- Conseguir visualizar o tamanho do log de dados
- Visualizar os dados do período desejado
- Salvar os dados do período desejado
- Sair da aplicação com sucesso
- Conferir exibição do menu de seleção na terminal
- Digitar a opção 1
- Verificar status da conexão
- Digitar a opção 2 para visualizar o tamanho do log de dados
- Digitar a opção 3 para visualizar os dados do log
- Digitar a opção 4 para salvar os dados do log
- Digitar a opção X para encerrar a aplicação