



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

Alysson José Mendes Borba

Estação Meteorológica Inteligente: Entrega final

Florianópolis
2022

Alysson José Mendes Borba

Estação Meteorológica Inteligente: Entrega final

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

O presente relatório detalha a modelagem e desenvolvimento do projeto aplicado da Estação Meteorológica. O objetivo principal do projeto é otimizar o consumo de energia da estação utilizando técnicas de aprendizado de máquina para aprimorar o tráfego de dados oriundos da estação. São apresentados três *softwares* para diferentes plataformas (computador, *smartphone* e software embarcado), todos utilizando a linguagem C++. O microcontrolador utilizado é o Raspberry Pi, aliado a um sensor de temperatura.

Palavras-chave: C++, estação meteorológica, microcontrolador, aprendizado de máquina, temperatura.

LISTA DE FIGURAS

Figura 1 – Fluxograma do <i>Software</i> do Computador.	6
Figura 2 – Diagrama de classes do projeto – Atualizado (Entrega 2).	7
Figura 3 – Fluxograma do <i>Software</i> Embarcado.	9
Figura 4 – Valores de temperatura para treinamento de modelo.	12
Figura 5 – Erro ao gerar modelo TensorFlow para microcontroladores.	12
Figura 6 – Aplicativo de recepção de dados.	14
Figura 7 – Utilização de memória no sistema embarcado.	16

SUMÁRIO

1	INTRODUÇÃO	5
1.1	OBJETIVOS	5
1.1.1	Objetivos específicos	5
2	DESENVOLVIMENTO	6
2.1	SOFTWARE DO COMPUTADOR	6
2.1.1	Diagrama de Classes	6
2.2	PLATAFORMA DE DESENVOLVIMENTO	8
2.2.1	Sensor de Temperatura	8
2.3	SOFTWARE EMBARCADO	8
2.3.1	Entrega 2 – 30/11/2022	9
2.3.1.1	Atualização da classe ClockCalendar	10
2.3.1.2	Implementação de arquivo para escrita e leitura dos logs	10
2.3.1.3	Tratamento dos dados recebidos	10
2.3.1.4	Criação das classes Iniciar, Menu e Embarcado	10
2.3.1.5	Criação e modificação de algumas funções da classe Fila e Node	11
2.3.1.6	Utilização do sensor interno de temperatura para coleta de dados	11
2.3.1.7	Treinamento do modelo de <i>Machine Learning</i>	12
2.4	SOFTWARE DO SMARTPHONE	13
2.4.1	Entrega 3 - 07/12/2022	13
2.5	PLANO DE TESTES	14
2.5.1	Entrega 4 - 14/12/2022	15
2.5.1.1	Análise de Desempenho	15
3	CONCLUSÃO	17

1 INTRODUÇÃO

É bastante comum que sistemas de monitoramento enviem dados continuamente, independentemente do valor a ser enviado. Em alguns casos, entretanto, não é necessário que esse envio seja feito caso os valores sejam redundantes, uma vez que isso pode ocasionar um uso excessivo de memória bem como diminuição na autonomia do sistema, já que muita energia é consumida para o envio dos dados.

Nesse contexto, técnicas de aprendizado de máquina (*machine learning*) podem ser implementadas visando aperfeiçoar tais sistemas, uma vez que, com base em um número de medições, é possível gerar um modelo que preveja o próximo dado. Dessa forma, é possível comparar um novo dado com o modelo previsto, e, apenas se o novo valor estiver fora de uma margem de erro, o dado é enviado para um servidor.

A aplicação proposta nesse trabalho é uma estação meteorológica que coletará dados de temperatura. Em seguida, será feita uma comparação entre o dado lido e o previsto, e apenas se o novo valor lido estiver divergente e fora da margem de erro, o novo dado será enviado para o servidor.

O presente relatório visa documentar as etapas de modelagem e implementação do código da estação meteorológica de acordo com as especificações solicitadas.

1.1 OBJETIVOS

Este trabalho tem como objetivo geral a implementação de um projeto de estação meteorológica, utilizando técnicas de aprendizado de máquina e programação em C++, visando o aprimoramento da autonomia de bateria.

1.1.1 Objetivos específicos

- Exercitar a modelagem orientada a objetos visando implementação em C++;
- 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.

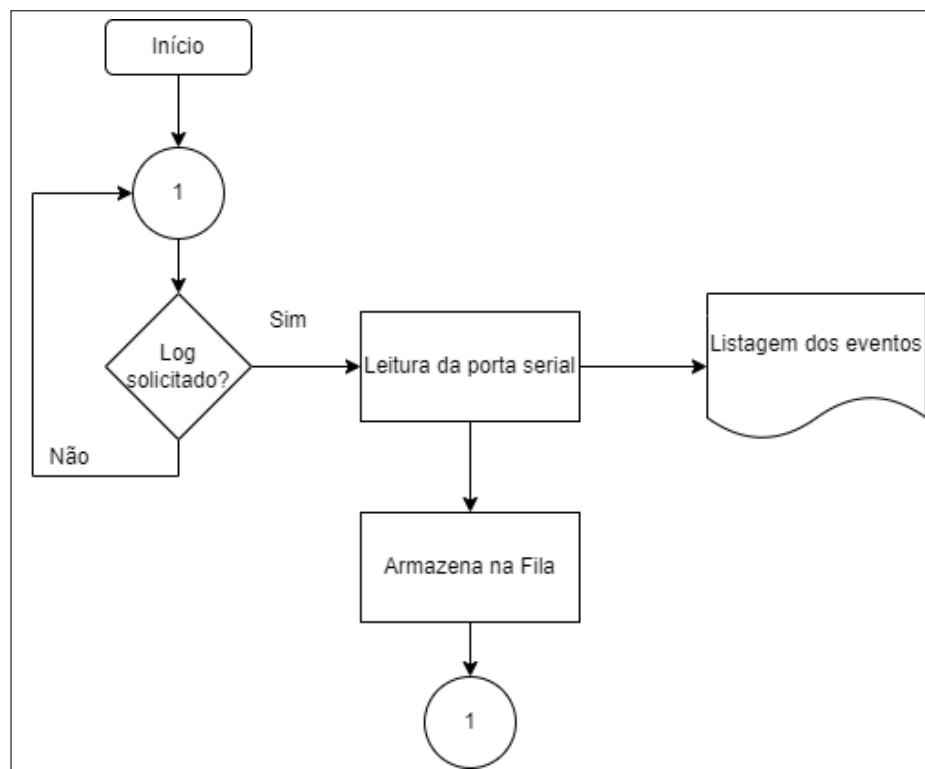
2 DESENVOLVIMENTO

O projeto é dividido em três diferentes *softwares*, sendo um para o microcontrolador, um para o computador e, por fim, um para o *smartphone*.

2.1 SOFTWARE DO COMPUTADOR

O *software* do computador é responsável por listar todos os eventos ocorridos em um determinado intervalo de datas. Para isso, é utilizado a comunicação com a porta serial (com cabo) para recebimento das informações. O fluxo do código é mostrado na Figura 1.

Figura 1 – Fluxograma do *Software* do Computador.



Fonte: Elaborado pelo autor (2022).

2.1.1 Diagrama de Classes

O diagrama de classes, mostrado na Figura 2, explicita todas as classes, atributos e métodos utilizados para o desenvolvimento do *software* e é explicado a seguir.

As classes *Clock* e *Calendar*, que já haviam sido criadas em aulas anteriores, bem como a classe filha *ClockCalendar*, que herda ambas, foram atualizadas para implementar o armazenamento da hora e data atuais, tendo em vista que as classes

Por fim, a classe `mySerial` foi utilizada visando implementar a comunicação com a porta serial, tanto para transmitir como para receber os dados. Para isso, são definidos alguns atributos como nome da porta serial (`deviceName`), e a velocidade de comunicação (`baud`).

2.2 PLATAFORMA DE DESENVOLVIMENTO

A implementação do sistema será realizada utilizando a plataforma Raspberry Pi 3 Model B. A placa possui entradas micro USB (para alimentação e transferência de dados e áudio) e HDMI (para a transferência de vídeo), além de interfaces para redes sem fio, como Bluetooth e Wi-Fi. Essas características de facilidade de interfaceamento com os dados fazem da Raspberry Pi uma alternativa compacta e de baixo custo para o projeto proposto. Existem diversas versões do microcomputador.

A Raspberry Pi roda o sistema Linux e, por isso, a programação, que pode ser realizada em diversas linguagens, é semelhante à estrutura da programação para PC. Nesse projeto, será utilizada a linguagem C++, visando pôr em prática os conceitos aprendidos em sala de aula.

O microcomputador Raspberry Pi 3 Model B possui 40 pinos, entre eles pinos de GPIO (*General Purpose Input/Output*, pinos de alimentação, e comunicação UART e I2C. Dessa forma, a Raspberry se torna um microcontrolador robusto, com capacidade computacional considerável e de baixo custo.

2.2.1 Sensor de Temperatura

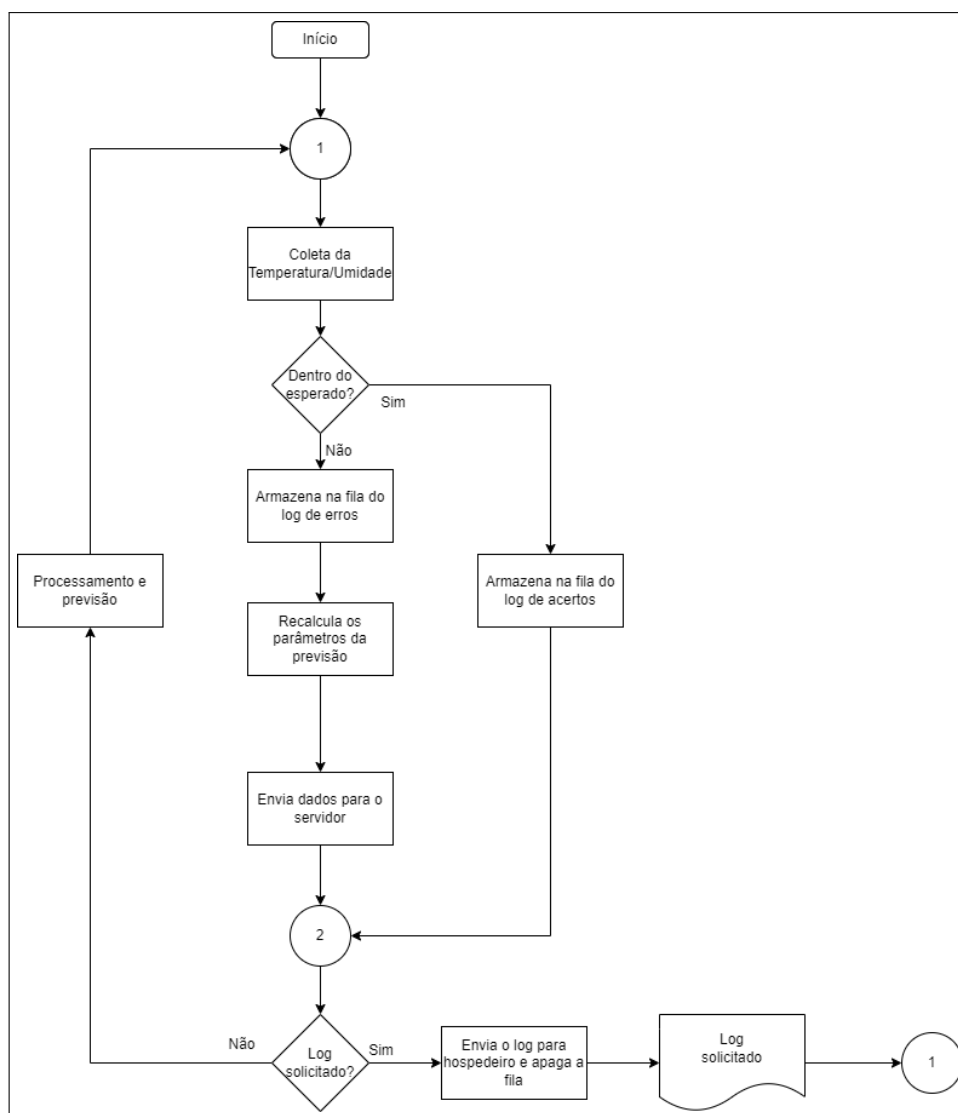
O sensor de temperatura utilizado para a coleta da grandeza foi o sensor interno da Raspberry Pi, que monitora a temperatura do processador. Mais detalhes sobre a implementação são dados na seção 2.3.1.6.

2.3 SOFTWARE EMBARCADO

O *software* do microcontrolador é responsável por fazer a leitura dos sensores, armazenar os *logs* de certos e errados, enviar para o servidor as ocorrências fora da margem padrão e enviar todos os *logs* para o *software* hospedeiro (computador e *smartphone*) quando solicitado.

Essa etapa do projeto segue o fluxograma apresentado na Figura 3.

Inicialmente, é realizada a coleta da grandeza a ser medida, que nesse caso é a temperatura. Através dos métodos de aprendizado de máquina do *framework* TinyML, será realizado o treinamento do modelo para identificar os padrões e prever os próximos valores de temperatura. Caso a nova temperatura coletada esteja dentro do esperado, então esse valor é armazenado em uma fila de "acertos". Caso contrário, o valor é armazenado na fila de erros, os valores dos parâmetros de previsão são

Figura 3 – Fluxograma do *Software* Embarcado.

Fonte: Elaborado pelo autor (2022).

recalculados e, em seguida, enviados ao servidor. O programa fica na espera da solicitação do *log* por parte dos hospedeiros. Quando o *log* for solicitado, então os dados são enviados para o hospedeiro por meio da porta serial.

O *software* do sistema embarcado será melhor explorado nas próximas etapas do projeto.

2.3.1 Entrega 2 – 30/11/2022

Para a entrega 2, os *softwares* foram atualizados de modo a implementar, de fato, o *software* na Raspberry Pi 3 Model B. Assim, as principais mudanças são abordadas a seguir.

2.3.1.1 Atualização da classe ClockCalendar

A classe *ClockCalendar*, que anteriormente tinha sido adaptada para salvar sempre o horário atual, foi reajustada para armazenar o horário atual apenas na função de leitura do sensor. No *software* do computador, ao recriar a Fila lida na memória, são lidos os horários nas quais as leituras foram realizadas, sejam por meio da porta serial ou por meio de um arquivo.

Essa funcionalidade foi alcançada utilizando sobrecarga de funções, de modo que foram criados construtores com parâmetros distintos. O construtor *ClockCalendar()* é chamado na leitura do sensor, para armazenar os dados na fila. Já o construtor *ClockCalendar(int, int, int, int, int, int, bool)*, com os parâmetros, é chamado na formação da Fila no software do computador, apenas lendo os dados. Os parâmetros passados são então mapeados para as variáveis de data e hora.

2.3.1.2 Implementação de arquivo para escrita e leitura dos logs

Como não é possível ler a porta da Raspberry Pi por meio de uma porta USB sem um conversor UART, foi criada uma solução para simular a transferência dos dados do software do sistema embarcado para o computador. Isso foi realizado através de um arquivo que é salvo pelo programa da Raspberry, que pode ser lido e interpretado pelo software do computador. No microcomputador, o arquivo é salvo no formato *[id,dd,mm,aa,hh,mm,ss,temp]*, uma linha para cada valor (cada nodo), armazenando assim todos os dados salvos na Fila.

A solução com a porta serial foi implementada, mas não pôde ser testada devido à falta do componente conversor USB-UART.

2.3.1.3 Tratamento dos dados recebidos

No *software* do computador, os dados recebidos (seja pela serial ou pelo arquivo) os dados são lidos no formato *string*. Sendo assim, foi necessário implementar funções para dividir a *string* lida, converter o tipo de dado (inteiro para id, dia, mês, ano, hora, minuto e segundo, e float para o valor de temperatura).

Após convertidos, os valores são armazenados na memória, utilizando a classe Fila. É possível, então, filtrar os dados pelo dia de cada registro.

2.3.1.4 Criação das classes Iniciar, Menu e Embarcado

Visando a utilização da Raspberry Pi tanto como um computador (com acesso à entrada e saída pelo monitor e teclado), quanto como um sistema embarcado para coleta e processamento de dados, o programa foi dividido nas classes Iniciar, Menu e Embarcado. A classe Iniciar é uma classe abstrata, que possui duas funções virtuais puras. As classes Menu e Embarcado, herdam a classe Iniciar.

A classe Menu destina-se à utilização como um computador, no ambiente Linux. Nesse menu, são apresentadas opções de leitura dos sensores, listagem dos valores lidos e armazenados na fila, envio dos valores para a serial e a opção de liberar memória deletando todos os valores armazenados. Também foi adicionada a opção de gerar um arquivo com dados para treinamento da inteligência artificial. Nessa classe, a opção de enviar dados para a serial foi configurada para salvar o arquivo com as informações

A classe Embarcado possui um construtor que inicializa o programa, fazendo a leitura do sensor em um *loop* infinito. Foi utilizada a biblioteca `<wiringPi.h>`, juntamente com a configuração necessária no compilador, para mapear os pinos digitais. Dessa forma, um pino digital é utilizado para ler um botão que indicará que os dados devem ser enviados para a porta serial. Caso um valor fora da margem esperada seja lida, esse valor é impresso na tela, e a fila de valores errados é apagada.

As classes foram implementadas utilizando o conceito de polimorfismo, sendo no arquivo principal declarado um ponteiro do tipo Iniciar, que, a depender da finalidade, inicializa a classe Menu ou Embarcado.

O diagrama de classes atualizado é visto na Figura 2.

2.3.1.5 Criação e modificação de algumas funções da classe Fila e Node

Foram corrigidos alguns *bugs* de *segmentation fault* que estavam sendo ocasionados ao apagar os dados da Fila. Para isso, foi necessário desfazer a herança da classe ClockCalendar, deixando apenas na forma de agregação.

Além disso, alterações foram feitas para permitir a inicialização de nodos com data e hora previamente configurados (e não com data e hora atuais) para possibilitar a funcionalidade da subseção 2.3.1.1.

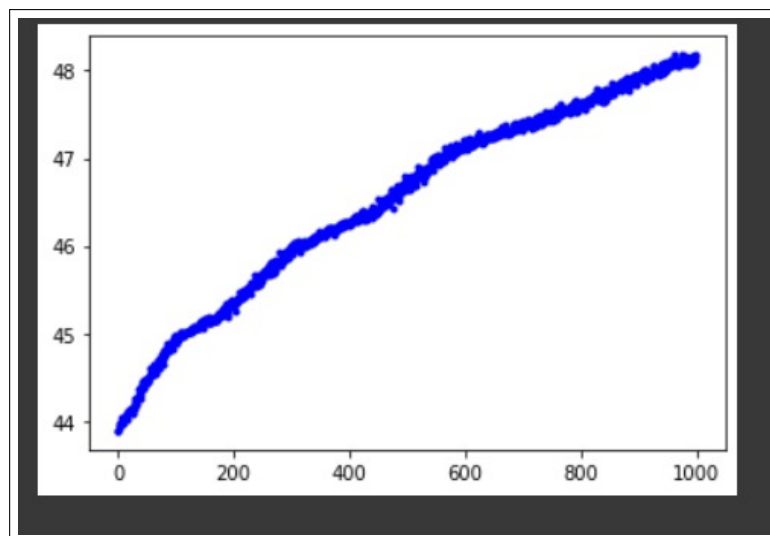
As funções *enviaFirst()* e *removeAll()* foram implementadas.

2.3.1.6 Utilização do sensor interno de temperatura para coleta de dados

A plataforma de desenvolvimento Raspberry Pi possui um sensor interno de temperatura, que monitora a temperatura da CPU. Esse sensor possui uma leitura máxima de 85 °C. É possível acessar esse sensor através da leitura direta dos registradores do processador. Existem diversos códigos na Internet que mostram como isso pode ser feito. Um deles foi utilizado (arquivo temperatura.cpp) e adaptado para retornar o valor lido por meio da função *leituraTemperatura()*.

Para gerar o arquivo de dados para treinamento do modelo no TinyML, foi feita uma média de 100 valores, realizando a aquisição de 1000 valores. No entanto, por se tratar de um sensor interno, o valor de temperatura aumenta enquanto o código está sendo rodado. A Figura 4 mostra o aumento da temperatura interna para realização das 1000 leituras em sequência.

Figura 4 – Valores de temperatura para treinamento de modelo.



Fonte: Elaborado pelo autor (2022).

Durante picos de processamento, foram detectados valores de até 50 °C.

2.3.1.7 Treinamento do modelo de *Machine Learning*

Foi implementada a função *gerarDadosModelo()*, que realiza a leitura da temperatura e armazena apenas os valores de temperatura em um arquivo. Os dados foram aplicados no *framework* TinyML, através do Google Colab.

As etapas foram corretamente seguidas e o modelo foi treinado. Entretanto, ocorreu um erro no momento de gerar as previsões do modelo TensorFlow, impossibilitando a geração do modelo. A Figura 5 mostra o erro apresentado.

Figura 5 – Erro ao gerar modelo TensorFlow para microcontroladores.

```
1. Predictions

# Calculate predictions
y_test_pred_tf = model.predict(x_test)
y_test_pred_no_quant_tflite = predict_tflite(model_no_quant_tflite, x_test)
y_test_pred_tflite = predict_tflite(model_tflite, x_test)

AttributeError                                Traceback (most recent call last)
<ipython-input-60-f7a48e06ea98> in <module>
      1 # Calculate predictions
      2 y_test_pred_tf = model.predict(x_test)
----> 3 y_test_pred_no_quant_tflite = predict_tflite(model_no_quant_tflite, x_test)
      4 y_test_pred_tflite = predict_tflite(model_tflite, x_test)

<ipython-input-51-a7bad81e8a15> in predict_tflite(tflite_model, x_test)
      7 # Initialize the TFLite Interpreter
      8 interpreter = tf.lite.Interpreter(model_content=tflite_model,
----> 9                                     experimental_op_resolver_type=tf.lite.experimental.OpResolverType.BUILTIN_REF)
     10 interpreter.allocate_tensors()
     11

AttributeError: module 'tensorflow_api.v2.lite.experimental' has no attribute 'OpResolverType'
```

Fonte: Elaborado pelo autor (2022).

O mesmo erro foi apresentado no exemplo "Hello World", da própria TensorFlow. Como o erro não foi corrigido, não foi possível exportar o modelo de aprendizado de máquina para a Raspberry Pi.

2.4 SOFTWARE DO SMARTPHONE

O *software* do *smartphone* realizará a mesma tarefa do *software* do computador, isto é, receber e armazenar os logs quando solicitado. Para isso, será utilizada a plataforma Android Studio.

Para programar para Android em C++, no Android Studio, é necessário instalar o Android Native Development Kit (NDK), que é um conjunto de ferramentas que permite usar código C e C++, além de oferecer bibliotecas da plataforma para gerenciar atividades nativas e acessar componentes de dispositivos físicos, como sensores e entrada por toque.

Idealmente, seria realizada a comunicação por meio de uma interface WiFi ou Bluetooth. No entanto, como essa etapa não será implementada, serão utilizados dados sintéticos para representar as informações obtidas do controlador proposto. Nesse projeto, o *software* será construindo simulando uma conexão Bluetooth.

Mais detalhes sobre o *software* para celular serão apresentados nas etapas posteriores do projeto.

2.4.1 Entrega 3 - 07/12/2022

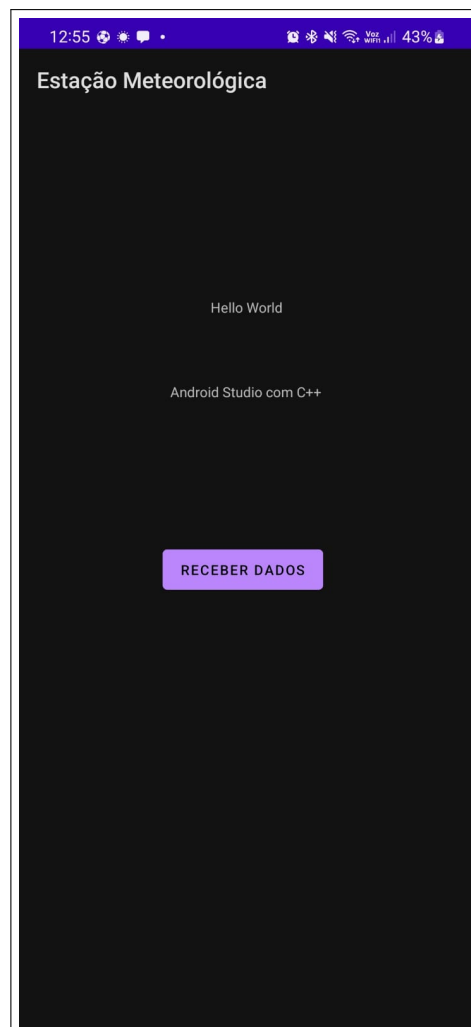
Para a terceira entrega do projeto, foi desenvolvida uma aplicação para o *smartphone* utilizando o *software* Android Studio. O aplicativo consiste em apenas uma tela, com uma mensagem de "Hello World" e um botão para recepção dos dados. A tela do aplicativo é vista na Figura 6.

O aplicativo ainda não está funcional, e o botão não executa ações, tendo em vista que o objetivo desta etapa era entender e testar o funcionamento do *software* Android Studio, utilizando a linguagem C++.

A utilização de C++ é apenas uma parte da programação, uma vez que o aplicativo em si é feito na linguagem Java. Isso é realizado através da JNI (Java Native Interface), que permite que métodos nativos sejam chamados durante a execução do código Java, o que é útil quando alguma funcionalidade não pode ser alcançada utilizando apenas Java. No código em C++, a JNI é inserida como uma biblioteca e o método é chamado no código Java.

O arquivo .apk, bem como os código-fonte, podem ser encontrados no repositório do GitHub https://github.com/alyssonmendes/estacao_meteorologica/.

Figura 6 – Aplicativo de recepção de dados.



Fonte: Elaborado pelo autor (2022).

2.5 PLANO DE TESTES

O projeto desenvolvido pode ser testado utilizando uma Raspberry Pi 3B e um computador. Na Raspberry Pi, o arquivo a ser compilado e executado é o `raspberry.cpp`.

Como mencionado anteriormente, o *software* da Raspberry Pi pode ser executado de duas formas diferentes através das classes Menu e Embarcado. Para acompanhar a leitura dos dados no sistema embarcado, deve-se comentar a linha de código que inicializa a leitura em loop infinito, deixando como mostrado abaixo.

```
init = new Menu();  
//init = new Embarcado();
```

Nesse menu, é possível Iniciar leitura do sensor, Exibir lista de acertos, Exibir lista de erros, Enviar dados pela serial, Gerar arquivo para o modelo do TensorFlow e Apagar todos dados das filas. Ao executar a primeira opção, são feitas 1000 leituras do

sensor, que são armazenadas nas filas de "certo"(dentro do previsto) e "errado"(fora do previsto). As listas podem ser consultadas nas opções de exibir.

Para executar o código no modo sistema embarcado, deve-se comentar a linha de inicialização do Menu, deixando apenas a classe Embarcado. Nesse modo, a leitura do sensor de dará de maneira infinita, e a Raspberry aguarda o pressionar de um botão para enviar os dados para a serial.

Vale salientar que para executar o arquivo `raspberrypi.cpp` é preciso especificar, durante a compilação, que a biblioteca `wiringPi` está sendo utilizada, através do comando `g++ -o raspberrypi raspberrypi.cpp -l wiringPi`.

No *software* do computador, cujo arquivo principal é o `computador.cpp`, é possível receber os dados (através da leitura de um arquivo ou da serial) e filtrar os dados recebidos por data. A leitura de dados pela serial não foi testada devido à falta de um conversor USB-UART.

Até o momento, o *software* do *smartphone* não executa ações e por isso não há um plano de testes definido.

2.5.1 Entrega 4 - 14/12/2022

2.5.1.1 Análise de Desempenho

O *software* mais complexo desenvolvido nesse projeto foi o do sistema embarcado. Para coleta dos dados e utilização do GPIOs, foi necessário incluir a biblioteca `WiringPi`. Além disso, por ser implementado de duas maneiras diferentes, utilizando polimorfismo, a Raspberry Pi foi utilizada de duas formas distintas, porém com o mesmo objetivo de coleta de dados pelo sensor.

A utilização do sensor de temperatura interno para simular o sensor de temperatura da estação implicou em alguns problemas de utilização e treinamento do modelo, uma vez que ao iniciar o executável, o processamento aumenta, elevando as leituras de temperatura. Isso impacta diretamente o treinamento do modelo do TensorFlow.

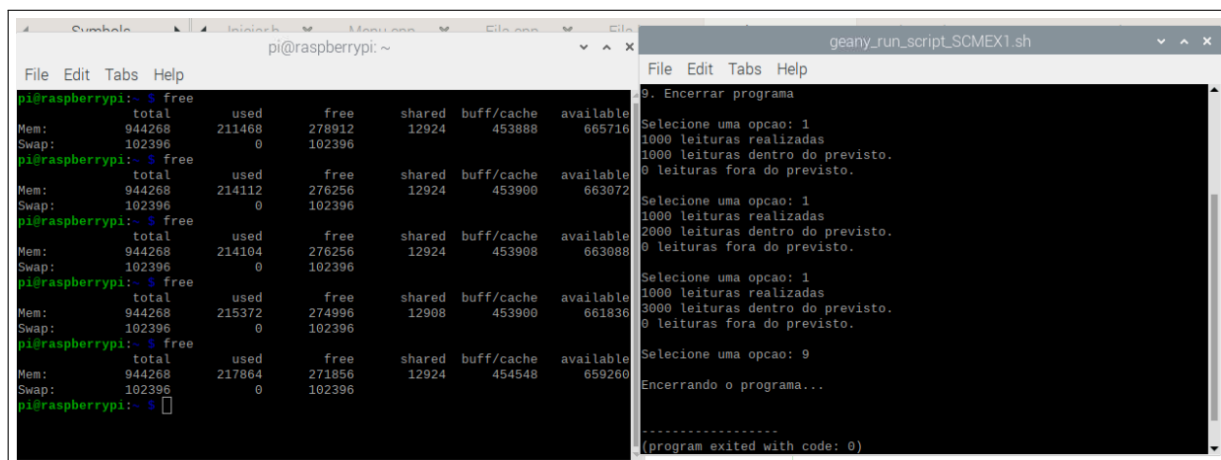
O erro de compilação do TensorFlow não foi corrigido, impossibilitando o treinamento do modelo e aplicação no *software* da RaspberryPi.

A utilização de memória da Raspberry Pi é bem administrada uma vez que é dado controle ao usuário de limpar os registros salvos quando preferir, além de limpá-los sempre que os dados são salvos no arquivo ou enviados pela porta serial. Além disso, a utilização e chamada de destrutores faz com que dados não fiquem "perdidos" na memória. Durante e após a finalização do código, todos os ponteiros criados são destruídos.

Na Figura 7, pode-se observar o uso de memória ao acrescentar leituras do sensor e o uso de memória da Raspberry Pi obtido através do comando `"free"`.

Não foi possível obter dados numéricos precisos do desempenho do código de-

Figura 7 – Utilização de memória no sistema embarcado.



The image shows a terminal window on a Raspberry Pi. The left pane displays the output of the 'free' command multiple times, showing memory usage statistics. The right pane shows a script execution with a menu and user input.

```
pi@raspberrypi:~$ free
total        used        free      shared  buff/cache   available
Mem:      944268      211468      728912       12924      453888      665716
Swap:      102396           0       102396
pi@raspberrypi:~$ free
total        used        free      shared  buff/cache   available
Mem:      944268      214112      726256       12924      453900      663072
Swap:      102396           0       102396
pi@raspberrypi:~$ free
total        used        free      shared  buff/cache   available
Mem:      944268      214104      726256       12924      453908      663080
Swap:      102396           0       102396
pi@raspberrypi:~$ free
total        used        free      shared  buff/cache   available
Mem:      944268      215372      724996       12908      453900      661836
Swap:      102396           0       102396
pi@raspberrypi:~$ free
total        used        free      shared  buff/cache   available
Mem:      944268      217864      721856       12924      454548      659260
Swap:      102396           0       102396
pi@raspberrypi:~$
```

```
geany_run_script_SCME1.sh
9. Encerrar programa
Selecione uma opcao: 1
1000 leituras realizadas
1000 leituras dentro do previsto.
0 leituras fora do previsto.
Selecione uma opcao: 1
1000 leituras realizadas
2000 leituras dentro do previsto.
0 leituras fora do previsto.
Selecione uma opcao: 1
1000 leituras realizadas
3000 leituras dentro do previsto.
0 leituras fora do previsto.
Selecione uma opcao: 9
Encerrando o programa...
-----
(program exited with code: 0)
```

Fonte: Elaborado pelo autor (2022).

vido ao gerenciamento feito pelo sistema operacional. Como visto na Figura 7, mesmo após acrescentar dados nas filas, a memória utilizada sofre pouca alteração.

O *software* do computador, por sua vez, é bem mais simples e necessita de poucos recursos do computador para ser executado.

3 CONCLUSÃO

Para a primeira entrega, foram desenvolvidos o *software* de recepção e um esboço do *software* que irá no sistema embarcado. Foi utilizada a classe Fila para armazenamento dos logs tanto na transmissão como na recepção. Cada Nodo da Fila, possui um código identificador, as informações de data e hora e o valor de temperatura lido. A comunicação é feita com a classe *mySerial*, que permite a leitura e escrita na porta serial.

Na segunda entrega do projeto, foram realizadas mudanças nas classes de modo a possibilitar a utilização na Raspberry Pi, além de serem implementadas as funcionalidades de leitura de temperatura utilizando o sensor interno da placa. O modelo de *machine learning* não pôde ser implementado devido a um erro no código do TensorFlow.

Na terceira entrega, foi desenvolvido um aplicativo exemplo no Android Studio, utilizando C++ através das ferramentas NDK e do interfaceamento com JNI. Além disso, foi apresentado um breve plano de testes, mostrando como o código deve ser compilado e executado.

Para a quarta e última entrega, foi demonstrado o funcionamento através de um vídeo e feita uma breve análise do desempenho dos softwares construídos. Os softwares se mostraram fluidos e com uma boa administração dos recursos de memória devido à utilização de destrutores e ao bom controle de recursos do sistema operacional.

A elaboração do projeto permitiu um aprofundamento nos conceitos de programação vistos em salas de aula, como a utilização de listas encadeadas, polimorfismo, utilização de classes, entre outros, além de permitir uma melhor familiarização com o sistema embarcado utilizado. O *software* do *smartphone* foi um bom ponto de partida para o início da programação para dispositivos Android, por permitir o uso da ferramenta Android Studio e do *framework* JNI. A utilização do TensorFlow para criação do modelo de *machine learning*, apesar de não ter sido completamente implementada devido ao erro da plataforma, foi útil por mostrar os passos necessários para a criação de um modelo de *machine learning* e aprendizado de máquina para sistemas embarcados.

O projeto pode encontra-se no repositório

https://github.com/alyssonmendes/estacao_meteorologica/ e o vídeo de funcionamento e apresentação pode ser acessado através do link

<https://www.youtube.com/watch?v=dC8IUVR-V1E>.